

Deliverable D2.1

SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment

WP 2

Project Acronym & Number:	SmartCLIDE – GA 871177
Project Title:	Smart Cloud Integrated Development Environment supporting the full-stack implementation, composition and deployment of data-centered services and applications in the cloud
Status:	Final
Dissemination Level:	Public
Authors:	UoM
Contributors:	ALL
Date:	31.08.2021
Revision:	1.0
Project website address:	https://smartclide.eu

Every effort has been made to ensure that all statements and information contained herein are accurate, however the SmartCLIDE Project Partners accept no liability for any error or omission in the same.

© 2020 Copyright in this document remains vested in the SmartCLIDE Project Partners.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871177

Project Consortium

Institut für angewandte Systemtechnik Bremen GmbH (ATB), Germany

INTRASOFT INTERNATIONAL SA (INTRA), Luxembourg

FUNDACION INSTITUTO INTERNACIONAL DE INVESTIGACION EN INTELIGENCIA
ARTIFICIAL Y CIENCIAS DE LA COMPUTACION (AIR), Spain

UNIVERSITY OF MACEDONIA (UoM), Greece

ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS (CERTH), Greece

X/OPEN COMPANY LIMITED (TOG), United Kingdom

ECLIPSE FOUNDATION EUROPE GMBH (ECLIPSE), Germany

WELLNESS TELECOM SL (WT), Spain

UNPARALLEL INNOVATION LDA (UNP), Portugal

CONTACT SOFTWARE GMBH (CONTACT), Germany

KAIROS DIGITAL, ANALYTICS AND BIG

DATA SOLUTIONS SL (KAIROS DS), Spain

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Change History

Version	Notes	Date
0.1	Creation of the document	14.03.2021
0.2	Research Approach for Service Discovery and Classification	15.05.2021
0.3	Research Approach for Service Creation and Composition	30.05.2021
0.4	Research Approach for Service Specification and Registry	30.06.2021
0.5	Research Approach for Cloud Applications Security, Maintainability, and Reusability	30.06.2021
0.6	Research Approach for Cloud Application Testing	30.07.2021
0.7	Research Approach for Code Generation and Pattern Selection Research Approach for Service Composition and AI	30.07.2021
0.8	Research Approach for Services and Workflows Deployment & CI/CD	30.07.2021
0.9	Internal review version	15.08.2021
1.0	Final version	31.08.2021

Executive Summary

The current document (D2.1) constitutes the first version of deliverable “SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment” of the SmartCLIDE project. The aim of the deliverable is two-fold: (a) to describe novel approaches for service discovery, creation, composition, and deployment; and (b) to provide details on the adoption of existing technological approaches for service discovery, creation, composition, and deployment. The first version of the deliverable will present: (a) an approach for service discovery and classification; (b) an initial version of service registry—the querying process will be finalized in the final version of D2.1; (c) a process for service creation and service specification; (d) an initial approach for source code generation—mostly focused on design pattern selection and skeleton creation; (e) a process for service composition; (f) an initial version for AI support in service composition; (g) an initial process for testing services and workflows; (h) an initial version for the security, maintainability, and reusability assessment of services and workflows; and (i) a process for service and workflow deployment.

Given the above, apart from the finalization of the aforementioned initial versions of approaches the final version of the deliverable will contribute: (a) service specification for runtime monitoring and verification; (b) source code autocomplete suggestions; (c) approaches for requirements assessment; and (d) approaches for service monitoring upon deployment.

Abbreviations

AAN	Artificial Neural Networks	GOOCA	Generic Object-Oriented Cryptographic Architecture
AMI	Advanced Metering Infrastructure	GNN	Graph Neural Network
AMQP	Advanced Messaging Queuing Protocol	GPU	Graphs Processing Unit
APAC	Architectural Pattern Application component	GRU	Gated Recurrent Unit
API	Application Programming Interfaces	HAC	Agglomerative Hierarchical Clustering
APIC	Architectural Pattern Inference component	LCC	Loose Class Cohesion
ASG	Abstract Syntax Graph	LCOM	Lack of Cohesion in Methods
AST	Abstract Syntax Trees	LDA	Latent Dirichlet Allocation
ATDD	Acceptance Test Driven Development	LOC	Lines of Code
BERT	Bidirectional Encoder Representations from Transformers	LSTM	Long Short Term Memory model
BDD	Behavior Driven Development approach	MDA	Model Driven Architecture
BLSTM	Bidirectional LSTM	MDD	Model Driven Development
BOW	Bag of words	MDE	Model Driven Engineering
BPMN	Business Process Model and Notation	ML	Machine Learning
CBO	Coupling Between Objects	MNB	Max Nested Blocks
CD	Continuous Delivery	MSMQ	Microsoft Messaging Queuing
CFG	Control Flow Graph	NIST	National Institute of Standards and Technology
CI	Continuous Integration	NLP	Natural Language Processing
CO2P2S	Correct Object-Oriented Pattern-based Programming System	NLTK	Natural Language Toolkit
CO2P3S	Correct Object-Oriented Pattern-based Parallel Programming System,	NSP	Next Sentence Prediction
CWE	Common Weakness Enumeration	NVD	National Vulnerability Database
DFG	Data Flow Graph	OAS	Open API Specification
DIT	Depth of Inheritance Tree	OMG	Object Management Group
DL	Deep-Learning	PVE	Parallel Video Encoder
DoA	Description of Action	QoS	Quality of Service
DPATool	Design Pattern Analysis Tool	QWS	Quality of Web Service
DPT	Design Pattern Tool	R	Recall
DSL	Domain Specific Language	RMI	Remote Method Invocation
ESB	Enterprise Service Bus	RNN	recurrent neural networks
FN	False Negatives	RSDL	Restful Service Description Language
FP	False Positives	SARD	Software Assurance Reference Dataset
GA	Grant Agreement	SAST	static application security testing
		SCM	Source Control Management
		SD	Services Deployment
		SDK	Software Development Kit
		SDLC	Software Development Lifecycle

D2.1 SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment

SOA	Service-oriented Architecture	UDDI	Universal Description, Discovery, and Integration
SOAP	Simple Object Access Protocol	URI	Uniform Resource Identifiers
SQLI	SQL Injection	VP	Vulnerability Prediction
SSA	Security-related Static Analysis	VPM	Vulnerability Prediction Model
SSAS	Security-related Static Analysis Subcomponent	WMC	Weighted Methods per Class
TCC	Tight Class Cohesion	WSDL	Web Services Description Language
TD	Technical Debt	WS-I	Web Services Interoperability
TDD	Test Driven Development	WSIL	Web Services Inspection Language
TN	True Negatives	WSMO	Web Service Modeling Ontology
TP	True Positives	WsRF	Web Service Relevancy Function
TSSS	Transformed Space Similarity Search	XSS	Cross-site Scripting

Table of Contents

1 Introduction 12

1.1 Document Purpose 12

1.2 Approach 12

1.3 Document Structure..... 16

2 Service Identification..... 17

2.1 Concepts 17

2.2 Gathering Service data 18

2.3 Proposed Models 22

3 Service Classification..... 26

3.1 Related Work..... 26

3.2 AI-algorithm selection..... 31

3.3 Case Study 32

3.4 Results 38

3.5 Conclusion..... 39

4 Service Registry 41

4.1 Introduction 41

4.2 Related work..... 42

4.3 Service extraction, integration, and negotiation techniques in multi-agent paradigm 42

4.4 Evaluation of proposed extraction methodology..... 45

4.5 Conclusions and Future work..... 48

5 Service Creation and Composition..... 49

5.1 Current Service Creation Practices..... 49

5.2 Composing Services with BPMN..... 54

5.3 Proposed Technological Solution for Service Composition..... 60

5.4 Proposed Technological Solution for Service Creation 61

6 Service Specification..... 64

6.1 Background 64

6.2 Related Work..... 65

6.3 Specify services from Web service registries..... 68

6.4 Specify Services from Code Repositories 68

6.5 Functional Service Specification Schema 69

6.6 Service Specification Validation 71

7 Code Generation 72

7.1 Code Generation Background 72

7.2	Template based code generation approach.....	74
7.3	Evaluation of generated systems	77
7.4	Conclusions	79
8	Design Pattern Selection.....	80
8.1	Design Patterns Selection and Application	80
8.2	Architectural Patterns Selection and Application.....	84
8.3	Security Patterns Selection and Application	90
9	Service Composition and AI	103
9.1	Workflow Functionality Summarization from BPMs	103
9.2	Service Composition Autocomplete Suggestions.....	107
10	Cloud Application Testing	113
10.1	Cloud Testing vs Testing a cloud.....	113
10.2	Types of Testing	113
10.3	Automated Performance Testing.....	115
10.4	Integrating Performance Testing into Development Workflow	116
10.5	Behaviour Driven Development.....	116
11	Cloud Applications Security, Maintainability, and Reusability	118
11.1	Maintainability Assessment	118
11.2	Reusability Assessment.....	121
11.3	Security Assessment.....	122
12	Services and Workflows Deployment & CI/CD.....	155
12.1	Innovative Approach to Enable Agnostic CI/CD tools	155
13	Conclusions.....	160
	References.....	161

List of Figures

Figure 1: SmartCLIDE research problems identification	13
Figure 2: WSDL service specification [3]	18
Figure 3: QWS dataset Information	19
Figure 4: Example of variety of service responses	20
Figure 5: Software Features	21
Figure 6: Online service discovery model using third-party search APIs	23
Figure 7: Local service discovery using Crawler and internal service registry	24
Figure 8: Web services ontology [30]	27
Figure 9: Common categories/ Label for service classifiers	29
Figure 10: Service classifier possible features	30
Figure 11: Related AI techniques in service classification	31
Figure 12: Proposed classification Model based on WSDL	32
Figure 13: Proposed classification Model based on REST services	33
Figure 14: Agglomerative Hierarchical Clustering result for service categories	35
Figure 15: Word cloud	36
Figure 16: Bar plot for clustered data	37
Figure 17: Pipeline of the service extraction process	43
Figure 18: List of Data sets obtained from websites	45
Figure 19: Valid entries per repository	46
Figure 20: Valid entries from Docker Hub	46
Figure 21: Missing values for the data set: programmableweb	47
Figure 22: Missing values for the data set: Service code repositories	47
Figure 23: Missing values for data set: DockerHub	48
Figure 24: GitLab repository creation	49
Figure 25: Jenkins main menu	50
Figure 26: Jenkins new Item	51
Figure 27: Jenkins Job General	51
Figure 28: Jenkins Job Build Triggers	52
Figure 29: Jenkins Job Pipeline Script	53
Figure 30: GitLab web-hook configuration	53
Figure 31: Example process that uses a Script Task as the first one in order to initialize some useful variables	60
Figure 32: Example business process in the online editor of JBPM Business Central	61
Figure 33: Process Transformation for Service Creation in SmartCLIDE	62
Figure 34: Example landing page	63
Figure 35: Web Service Stack in Restful and Soap Services [18][155]	65
Figure 36: Extractable features from most popular online sources	67
Figure 37: Service Functional Specification Schema	70
Figure 38: Agent specification in file	74
Figure 39: Phase I of code generation process	75
Figure 40: Phase II of code generation process	75
Figure 41: Phase III of code generation process	76

Figure 42: Phase IV of code generation process	76
Figure 43: Agent organization specification file	77
Figure 44: Generated HTTP rest adapter	78
Figure 45: Java generated code from agent side	78
Figure 46: Negotiation Results	78
Figure 47: Check Point Flow Diagram [133]	93
Figure 48: Secure Logger Flow Diagram [133]	95
Figure 49: Authenticator Flow Diagram[133]	96
Figure 50: Session Pattern Flow Diagram [133]	96
Figure 51: Relationships between cryptographic design patterns [25]	98
Figure 52: Block diagram of a typical composition/pattern around the CHECK POINT pattern ... 99	
Figure 53: A high-level overview of the mechanism for selecting security patterns and suitable technologies	101
Figure 54: Example of code snippet corresponding to Basic Auth with Java API	102
Figure 55: High-level process of text summarization	103
Figure 56: Internal Architecture for the proposed Text Summarization service	104
Figure 57: Simplified input process	105
Figure 58: Input process with branching	106
Figure 59: Input process with two Gateways	106
Figure 60: Input process with a Timer	107
Figure 61: High level approach of Service Composition suggestions	108
Figure 62: DLE BPMN Suggestions Architecture	109
Figure 63: XML-BPMN simplified	110
Figure 64: DLE services dataset visual representation	111
Figure 65: TD Principal and Interest [31]	119
Figure 66: Contents of gitlab-ci.yml file for SonarQube analysis	120
Figure 67: Sequence to Sequence model Overview	126
Figure 68: The Attention overview	126
Figure 69: An example of Attention usage	127
Figure 70: Transformer overall architecture	127
Figure 71: The Transformer Encoder model (BERT)	128
Figure 72: Model output (JSON format) for the Cpp case study	133
Figure 73: Model output (JSON format) for the Java case study	134
Figure 74: High-level overview of the security analysis framework/platform – The Security-related Static Analysis Subcomponent (SSAS)	140
Figure 75: A screenshot of the HTTP Request submitted fro analyzing the security of an open-source Java project	148
Figure 76: The HTTP request that should be submitted for analyzing the selected open-source Python Project	151
Figure 77: Continuous Integration, Delivery and Deployment	155
Figure 78: Brief schema of a Workflow	158

List of Tables

Table 1: Microplanning per task of WP2 and relevant problems	14
Table 2: Available Public Datasets	29
Table 3: Dataset Information-1	38
Table 4: Dataset Information-2	39
Table 5: Case Study Result	39
Table 6: Basic BPMN Modelling Elements.....	54
Table 7: Events	58
Table 8: Review of Functional and Non-Functional Service Features	66
Table 9: Possible types of service specification.....	68
Table 10: NFRs supported by Architectural Patterns	89
Table 11: Requirements, Architectural and Design Patterns [132].....	97
Table 12: Mapping between Security Requirements, Security Patterns, and actual frameworks/libraries that can be used for their implementation into software	100
Table 13: Generic Characteristics of the Workflow	104
Table 14: Hyper-parameters of BERT base model.....	131
Table 15: Fine-tuning hyper-parameters.....	131
Table 16: Evaluation results.....	131
Table 17: Open-source and Commercial Static Code Analyzers.....	138
Table 18: Metric-based Security Properties supported by the SSAS.....	142
Table 19: Alerts-based Security Properties supported by the SSAS.....	142
Table 20: Recommended Security Characteristics (i.e., Requirements) to be used in the analysis performed by the SSAS.....	144
Table 21: The thresholds of the properties of the security model for analyzing Java projects	146
Table 22: The thresholds of the properties of the security model for analyzing Python projects.....	146
Table 23: The weights of the security model for analyzing Java projects	146
Table 24: The weights of the security model for analyzing Python projects.....	147
Table 25: The parameters of the HTTP Request that should be submitted for analyzing a specific software project using SSAS.....	148
Table 26: The parameters of the HTTP Request that were used for analyzing the selected open-source Java project.....	148
Table 27: The results of the analysis of the selected open-source Java project.....	149
Table 28: The parameters of the request that were submitted for analyzing the selected open-source Python Project	152
Table 29: The results of the analysis of the selected open-source Python project.....	152
Table 30: State of the Art of CI/CD tools	156

1 Introduction

1.1 Document Purpose

Deliverable “D2.1 *SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment*” is the outcome of WP2, produced cumulatively from all tasks of WP2. This deliverable is one of the core research deliverables of the SmartCLIDE project and documents all innovative approaches that have been introduced in the course of the project. The achieved innovation lies on either the proposal of novel approaches, or the innovative synthesis of existing technological approaches into automated processes that can aid in the development of service-oriented applications. This deliverable is horizontal in the sense that it covers innovations produced through the R&D cycles of all SmartCLIDE components specified in D1.4 “*SmartCLIDE Concept*”. The produced innovative approaches (as part of this deliverable) are going to be accompanied by research prototypes, serving as means of evaluation. The research prototypes will be fed to the SmartCLIDE framework IDE (WP3), which will in turn result in the Early (Task 4.2) and Full prototypes (Task 4.3). All software artefacts (research prototypes) have been kept in the consortium’s GIT repository.

1.2 Approach

The approach that we have used for implementing WP2 and guide the research activities into SmartCLIDE was the engineering cycles, as described in the design science framework [156]. According to Wieringa [156], every engineering problem can be treated as a 4-step process: (a) identifying the need and specifying the problem; (b) design the proposed solution; (c) evaluate the proposed solution; and (d) apply the solution. This deliverable aims at the first 3 steps of the approach in the sense that the application of the solution falls into WP5 “*Assessment of SmartCLIDE at pilot users*”. In this section, we focus on the 1st step identification of problems; whereas the 2nd (proposal) and the 3rd (evaluation) steps will be presented in the upcoming sections (2-13).

The identification of the targeted problems stemmed from the DoA (as described in the GA) and the requirements of SmartCLIDE (D1.2 “*Requirements Analysis*”). Next, each problem has been decomposed to simpler tasks that need to be fulfilled to solve the problems. The tasks are decomposed to two main categories:

- **technological tasks** aim to solve problems by reusing or adapting existing solutions. The advancement that technological tasks is the introduced level of automation, as well as the composition of existing solutions into processes.
- **research tasks** aim to solve problems that cannot be treated with existing solutions; thus, urge for novel algorithms, prototypes, and tools.

The outcomes of these tasks are organized into seven types:

- **reports** correspond to documents describing existing tools, repos, approaches, etc.;
- **datasets** correspond to collections of data points that will be stored in a repo;
- **schemas** correspond to documents describing the format of data stored in repos, or exchanged between components;
- **tool** corresponds to existing implementations that solve a practical problem;
- **approach** corresponds to novel research approaches (method, algorithm, etc.) for problem solving. They are expected to be linked to a publication;

- **draft prototype** corresponds to the proof-of concept implementation of novel approaches. This version of the implementation is used only for research purposes. We expect a low level of automation and no integration at this stage;
- **final prototype** corresponds to the functionally final version of the research prototype. This version is going to be handed to WP3. This version will be fully automated, no integration. This prototype is still considered as part of WP2, since can ripple changes in the research approach. In WP3 the specific prototype will be adopted to the platform, exhaustive testing will be performed.

In Figure 1, we map the problems identified in the DoA and requirements to the tasks of WP2, so as to guide the organization of this deliverable and the research activities.

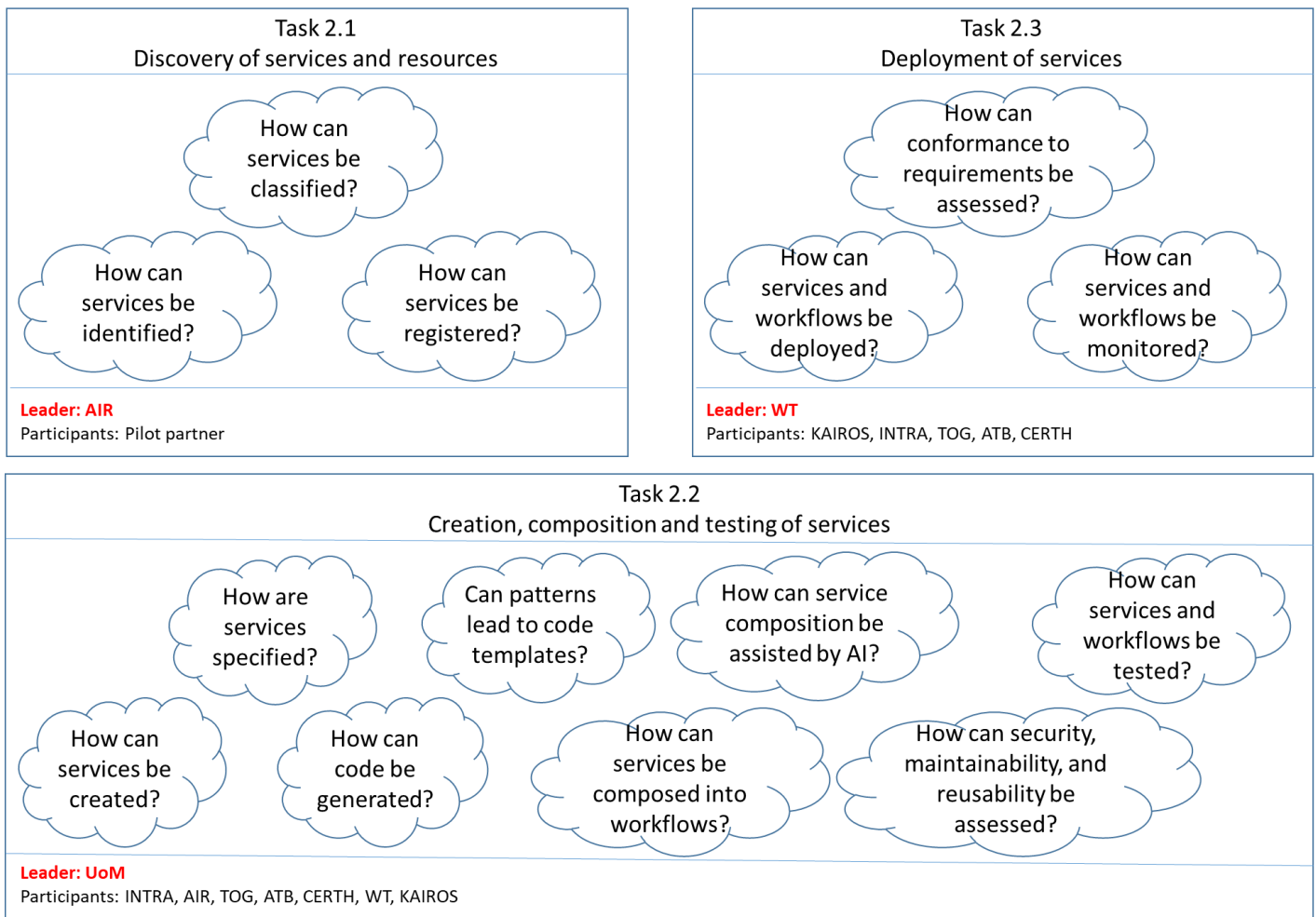


Figure 1: SmartCLIDE research problems identification

Upon problem identification, we proceeded to the assignment of responsibilities for answering each problem, and a microplanning approach was used for monitoring. The final microplanning (pertaining to the period of both D2.1 and D2.2) is presented in Table 1.

Table 1: Microplanning per task of WP2 and relevant problems

Task	Problem	Solution	Partner	Outputs
2.1	How can services be identified?	Research approach on getting services from classic registries Research approach on getting services from web pages Research approach on getting services from code repositories	AIR	Report
	How can services be classified?	Research approach on available datasets Research approach on classification model implementation		Approach Dataset Prototypes
	How can services be registered?	Research approach on registry service query Research approach for interfacing service registry		Approach Prototypes
2.2	How can services be created?	Technological Approach on the Integration of Version Control in SmartCLIDE Technological Approach for service creation in SmartCLIDE	INTRA UoM	Tool Approach Prototypes
	How can services be specified?	Technological Approach on Functional Service Specification Technological Approach on the Specification of Service Runtime Monitoring & Verification	AIR TOG	Schema
	How can code be generated?	Research Approach on Source Code Generation Research Approach on Autocomplete Suggestions	AIR	Approach Prototypes
	Can patterns lead to code templates?	Research Approach on Design Patterns Default Implementations Research Approach on Architectural Patterns Default Implementations Research Approach on Security Patterns Implementations	UoM ATB CERTH	Approach Prototypes
	How can services be composed into workflows?	Technological Approach on Service Composition Representation Using BPML Technological Approach on Service Composition (either Discovered or Created) Technological Approach for Mapping Services to Containers	UoM UoM WT	Tool Approach Prototypes

Task	Problem	Solution	Partner	Outputs
	How can service composition be assisted by AI?	Research Approach on Autocomplete Suggestions on Service Composition Research Approach on Workflow Context Identification	UoM ATB	Approach Prototypes
	How can services and workflows be tested?	Technological Approach for Coverage of Created Services Technological Approach for Unit and Acceptance Testing Research Approach for Automated Test Case Generation (Virtual User for Testing)	KAIROS UoM KAIROS	Tool Approach Prototypes
	How can security, maintainability, and reusability be assessed?	Research Approach for Security Assessment at Service Level Research Approach for Security Assessment at Workflow Level Research Approach for Maintainability Assessment at Service Level Research Approach for Maintainability Assessment at Workflow Level Research Approach for Reusability Assessment at Service Level Research Approach for Reusability Assessment at Workflow Level	CERTH CERTH UoM	Approach Prototypes
2.3	How can services and workflows be deployed	Technological Approach on Deployment and Orchestration Tools Technological Approach on Management Tools Research Approach on Continuous Delivery	WT WT KAIROS	Tool Approach Prototypes
	How can conformance to requirements be assessed?	Research Approach on Cost Analysis Research Approach on Scalability Assessment	WT INTRA	Approach Prototypes
	How can services and workflows be monitored?	Technological Approach on Runtime Monitoring and Verification of Services Research Approach on Defining Sensors/Metrics for Security Monitoring Technological Approach on System Monitoring (Performance and QoS)	TOG CERTH ATB	Approach Prototypes

1.3 Document Structure

The rest of the core document is organized into 11 sections, each one corresponding to the lines of Table 1 (research problems). Sections 2, 3, 5, 6, 10, and 12 present approaches that are close to final while Sections 4, 7, 8, 9, and 11 are initial versions. In any case, all approaches/techniques are going to be finalized in the final version of this deliverable D.2.2 after having validated them with the pilot providers of the project. Finally, Section 13 concludes this deliverable by summarizing the main outcomes and provides an overview of the envisioned update for the final version, i.e., D.2.2.

The publications that have been achieved so far are reported as part of the Dissemination deliverable, although their material is used in this document. We note that the number of publications is expected to increase for the 2nd period of this task, since until now the research team were focused on the development of the approaches and research prototypes. In the 2nd period, when the experimentation with the pilots will be progressed, more complete/mature research works will be delivered.

2 Service Identification

The development of Service-oriented Systems can be divided into 4 steps: (a) Identifying system requirement and required tasks; (b) Finding service registries, providing a pool of services (Service discovery); (c) Classifying the discovered services to build a list of candidate services with the same functionality; and (d) Ranking the selected services based on service quality features (user scoring, response time, etc). In this section we focus on the second step, which is service discovery. The service discovery tactics adopted by service registries can be narrowed down into the following approaches:

- Keyword-based service discovery such as UDDI (Universal Description, Discovery, and Integration)
- Expand queries with relevant concepts extracted from lexical databases (e.g., WordNet)
- Semantic-based approaches [32][48][92]. The idea behind this approach is to annotate services for future use, in order to be able to search for them based on annotation rules (ontologies). There are standards used in semantic-based approaches such as WSMO (Web Service Modeling Ontology) [52], OWL-S[1], etc.
- AI-based approaches e.g., service classification.

2.1 Concepts

In this section we provide an overview of the available basic concepts and methods behind Service-oriented architecture. These concepts define the terms used throughout this document so as to avoid ambiguity.

- **Service:** A web service is a set of related application functions that can be invoked programmatically through the Internet. Web services allow businesses and customers to interact with each other by dynamically connecting and executing real-time transactions with minimal human interaction. Services are therefore self-contained, self-describing, modular applications that can be published, located, and invoked from the Web.
- **Service Meta-data:** A service is registered with its metadata, which include relevant information about it. This information can include the service name, type, input, output, description, and the service's endpoint.
- **Service Abstraction:** A service *abstraction* consists of an *abstract* of operations and parameters which help services act as black boxes.
- **Service Discovery:** In general, services are enriched with communicative metadata. The process of finding services and their endpoints using this metadata is called service discovery.
- **Service Registry:** A service that provides references to other services.
- **Restful** (Representational State Transfer): Restful appeared as a lightweight alternative for SOAP-based services. Restful services are designed to ease the discovery, composition, and building of community-driven services.
- **Web services standards:** some standards play key roles in Web services: UDDI (Universal Description, Discovery, and Integration), WSDL (Web Services Description Language), WSIL (Web Services Inspection Language), SOAP (Simple Object Access Protocol), and WS-I (Web Services Interoperability).
- **WSDL** is an open XML-based specification that describes the interfaces and instances of Web services on the network. It is extensible so that endpoints can be described regardless of the message formats or network protocols used to communicate. Enterprises can make WSDL documents

available to their Web services via UDDI, WSIL, or by disseminating the URLs to their WSDL via e-mail or Web sites.

- **SOAP** is an XML-based standard for the transmission of messages over HTTP and other Internet protocols. A lightweight protocol for information exchange in a decentralized and distributed environment, SOAP enables the binding and use of found Web services by defining a message path for addressing. SOAP can be used, for example, to query UDDI for Web services.
- **UDDI**: The UDDI specification defines open, platform-independent standards that allow companies to share information in a global company registry, find services in the registry, and define how they act together on the Internet. It separates web services into three colors.
 - White Pages: contact address and other identifiers.
 - Yellow Pages: industry categorization based on taxonomies
 - Green Pages: technical information on services provided by the companies themselves.

Its purpose is to be accessed by SOAP messages and to lead to WSDL documents, which describe the protocol requirements and the requested message formats to interact with the Web services of the catalogue of records.

- **WSIL**: WSIL is an open XML-based specification that defines a distributed service discovery method that provides references to service descriptions at the service provider's point of offerings, specifying how to check if Web services are available on a Web site. A WSIL document defines the locations on a Web site where Web service descriptions can be searched for. Since WSIL focuses on the discovery of distributed services, the WSIL specification complements UDDI by facilitating the discovery of services that are available on Web sites that may not yet be listed in a UDDI registry.

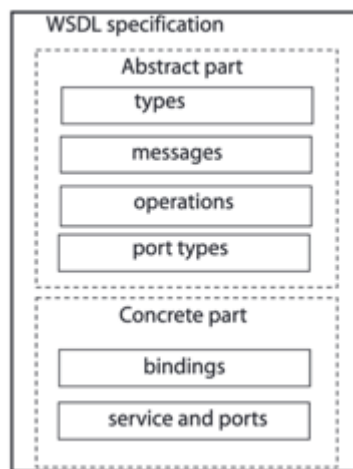


Figure 2: WSDL service specification [3]

2.2 Gathering Service data

This section focuses on discussing and investigating service data collection from web pages, code repositories, and classic registries.

2.2.1 Gathering services from web pages

The automatic gathering of service data and the challenges that this entails are described in this section. In general, a website includes names, descriptions, and service links. The service link contains service WSDL or a list of the service's methods, inputs and output.

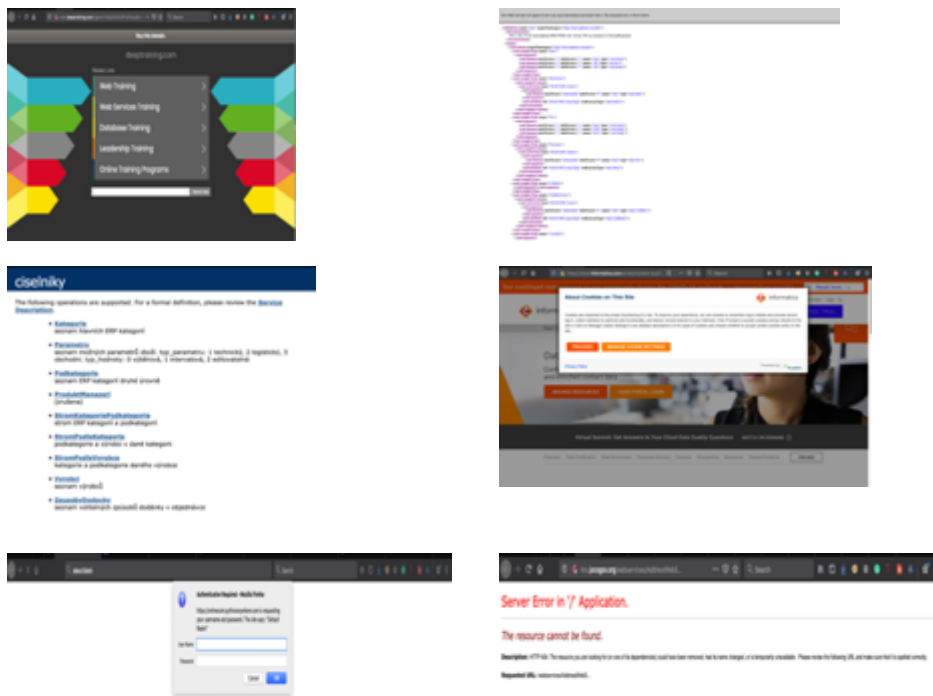
The [programmableweb¹](https://www.programmableweb.com/) offers a large API Directory (24,046), which provides web services based on the capabilities of Google, Facebook and Amazon. With this web service registry, we gained access to Service URLs, versions, source codes, SDK, following users, etc. However, only well-known services have full annotations. Although using programmableweb can be very useful, the literature has shown that such databases may stop being supported after a while; such is the case with <https://www.mashape.com/>. Another example is the QWS dataset that contains the following data:

```

RangeIndex: 2507 entries, 0 to 2506
Data columns (total 11 columns):
response_time      2507 non-null float64
availability       2507 non-null int64
throughput        2507 non-null float64
successability    2507 non-null int64
reliability       2507 non-null int64
compliance        2507 non-null int64
best_practice     2507 non-null int64
Latency           2507 non-null float64
Documentation     2507 non-null int64
service_name      2507 non-null object
wsdl_address      2507 non-null object
dtypes: float64(3), int64(6), object(2)
memory usage: 215.6+ KB
    
```

Figure 3: QWS dataset Information

In our experimentation, 314 pages service links of QWS dataset were tested. The main problem was that the web service links had a different types of responses. The majority of wsdl_address responses included XML or HTML pages.



¹ <https://www.programmableweb.com/api/>



Figure 4: Example of variety of service responses

In general, this form of collection creates noisy data. Therefore, it needs to be filtered out. Usually, cloud services are implemented using SOC standards such as WSDL, therefore a possible type of filtering will be to configure the search to collect files with UDDI, WSDL, and WADL extensions. A second solution would be to use the previous solution to collect WSDL documents and tags of services of interest through search engines. The above would aim to post-process the data obtained, and create a database of web services that can then be data-mined, for example to classify them or assign scores. However, service API (REST) URLs can support HTML pages response. Each service page can have its format and present service functions differently. To this end, it would be useful to use ML techniques such as TF-IDF to extract the relevance of keywords. In the service classification section of this document we investigated service data extraction and clustering to provide an informative dataset for future text classification. Moreover, it can be useful to combine both service collection and sorting approaches to generate a comprehensive service datasheet.

2.2.2 Gathering services from code repositories

A repository is a location for the mass storage of files which use version control systems to store multiple versions and track the changes of each file. Service trackers are a useful way to find already deployed and public solutions through web protocols. Moreover, it can be used in searching for image or container libraries-repositories, or to find services that are currently under development or ready to be deployed. For this purpose it is possible to use of the APIs of Github, GitLab, BitBucker, Source-forge, Launchpad, Google Source Repositories, etc., to collect code repositories, binaries and files. The projects on these platforms are usually adequately documented and generally come with description files and tags describing the functionality of the code. The big advantage is that the code hosted on these platforms is generally open-source, allowing more specific searches on specific service files or even sections of code.

An example of service-focused file collection would be to go through the contents of files and filter by

- keywords of certain SOCs
- by size
- number of followers
- number of forks
- number of stars and dates of update of certain files

or search for:

- files with .wsdl extensions
- sections of code describing services
- belonging to a specific organization

For example, <https://developer.github.com/v3/>, GitHub APIs can help to define software features. The following features can be obtained from GitHub APIs:

- User rate
- ReadMe file
- Software License
- vulnerability alerts
- Last commit date
- Requirement of software
- Number of issues
- Data or parameter Media type
- Number of pull request comments

Mentioned features are used for soft classifications which are different for each code repository. We have a manual annotation provided by [122] of 4,226 README file sections from 393 randomly sampled GitHub repositories. They have used manual labelling for their dataset and they have used supervised learning for multi-label classifier datasets. Moreover, we can use the number of issues, user rate, issue comments, pull request and last commit date for source code quality evaluation. However, to access and use this information for our advantage, code mining and source code clustering techniques are required. Another possible solution would be to look for service images or a set of ready-to-deploy services that make up more elaborate applications via containers, e.g. Docker Hub, Amazon Elastic Container Registry, JFrog Artifactory, and Harbor among others.

As with code repositories, applications are generally well documented and include descriptions, tags, and links to code repositories, and platforms similar to Docker Hub have a star rating system that indicates the level of popularity of the container.

Software meta data	Software name
	Software comment
	Software version
	Software description document
	Software License
Software code	Comment
	Source code
	Line of codes
	Size of codes
	Required library
	Language programming tokens
	Language programming tags
	First lines of code to extract libraries
	Structure level: page, class, function (this feature relates to second model that we will describe it later)

Figure 5: Software Features

2.2.3 Service Registries

A web service registry is a place where services are listed in order to be discoverable; therefore the Service Registry is closely related to the Service Discovery. The challenges related to Service Registry

and Discovery are as follows: “How to find appropriate services for on users’ needs from a large pool of available services” and “The client does not know which provider offers a suitable service”.

To discover the services, systems need a service registry:

1. First the client request is sent to search among the registered services.
2. A brief list of discovered services will be sent to the client as a response
3. The client selects a service from among the discovered services list, then requests the service detail from service providers.
4. The provider send the service’s details and information to the client
5. Based on the detailed information sent by the provider, the client will send their request to the target service.

Service registries are proposed to use an external registry with built-in service discovery, which is responsible for registering and unregistering a service instance in the service registry. Therefore, once the discovered services have been collected, they can be stored and classified. Towards that goal, SmartCLIDE may provide some internal services to enhance functionality. Hence, not only service metadata will be stored on databases (SQL/NoSQL), but also some Container Orchestration and Clustering Tools (e.g. Kubernetes, Mesos) may be needed. In general, these tools retrieve the status of all existing tasks from all running frames and generates DNS records for these tasks.

2.3 Proposed Models

In this section, we present the proposed models for service discovery in SmartCLIDE. SmartCLIDE service discovery will be based on metadata / source codes, and will draw results both online and locally.

2.3.1 Identify from Online Sources

There are popular online repositories that provide APIs allowing developers to make use of their data. Relying on this method, SmartCLIDE can get user data, after required preprocessing, send a proper request for repository APIs (e.g search API). The online discovery method can have the following steps (see Figure 6):

1. User can select default third party repositories
2. User send a query through a Natural Language Query Interface for searching Services
3. SmartCLIDE can be rewriting the user input query on the basis of the indexed popular search query or AI-based techniques
4. SmartCLIDE invokes Third-party search APIs
5. SmartCLIDE displays parsed and ranked results to Users

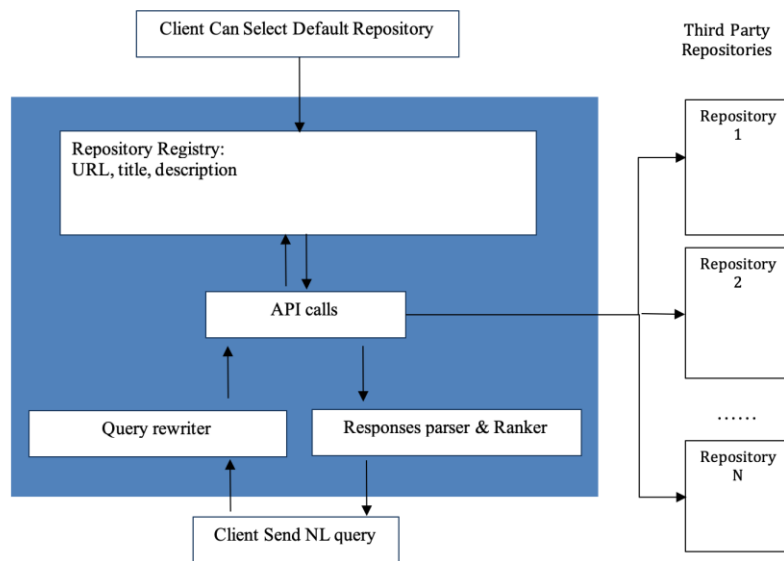


Figure 6: Online service discovery model using third-party search APIs

This method has the following benefits:

- These repositories usually return more recent, popular and available services.
- Gathering information from scratch is difficult. Under these circumstances, using existing repositories and their search APIs can save time.

However, there are some limitations for live service identification:

- Limitation to using API. For example, GitHub API requests using Basic Authentication or OAuth, thus limiting the client's requests to up to 5,000 per hour.
- Also, some available repositories do not have a search API due to web scraping.

Regarding these issues, a solution is to identify and gathering service data offline and update it regularly.

2.3.2 Identify service data locally (offline search)

The automated data-gathering imports data into the SmartCLIDE database and updates it frequently. Manually adding or updating data to the database can be made more efficient through tools/scripts that can update it automatically and quickly. Gathering offline service data can be categorized into three different forms: (a) Manual; (b) Semi-automated; and (c) Full-Automated methods.

- **Manual methods:** In this method, expert users extract and record service information manually. It requires a lot of time; however, the qualities of the results in accuracy and annotation level are high. These semantic annotation services help train algorithms to improve overall search.
- **Semi-automated methods:** In these methods, it is necessary to provide some information for machine extraction of the service.
- **Full-Automated methods:** This method automatically extracts all services from available sources. For this purpose, based on the repetition patterns of the services repository, a pattern-based script can be provided or machine learning models can be used from datasets.

One of the popular approaches is to build a service crawler that uses the APIs of existing search engines, where a number of levels of the service ontology of interest are chosen as keywords to

collect data on cloud services, IaaS, PaaS, SaaS, storage, infrastructure, online backup, web hosting, virtual desktop, virtual machine, software, APIs, etc.

The crawler collects possible entries from cloud services by continuously analysing search results from the indexes returned by search engines. However, a variety of service representations may exist due to the prior use of semi-automated methods.

For example, an approach would be to use a brute-force-based service search system to navigate the subspaces of a website using a dictionary of keywords to explore the subdomains. In this approach, the dictionary entries are gathered manually.

The demonstrated model can have the following steps (Figure 7):

1. The user sends a query through a natural Language Query Interface to search for Services
2. SmartCLIDE can rewrite or expand the user input query. Expanded queries can use relevant concepts extracted from lexical databases (e.g., WordNet) or domain ontologies [169].
3. The processed query will be sent to search engines for query service meta data
4. SmartCLIDE displays parsed and ranked results to Users

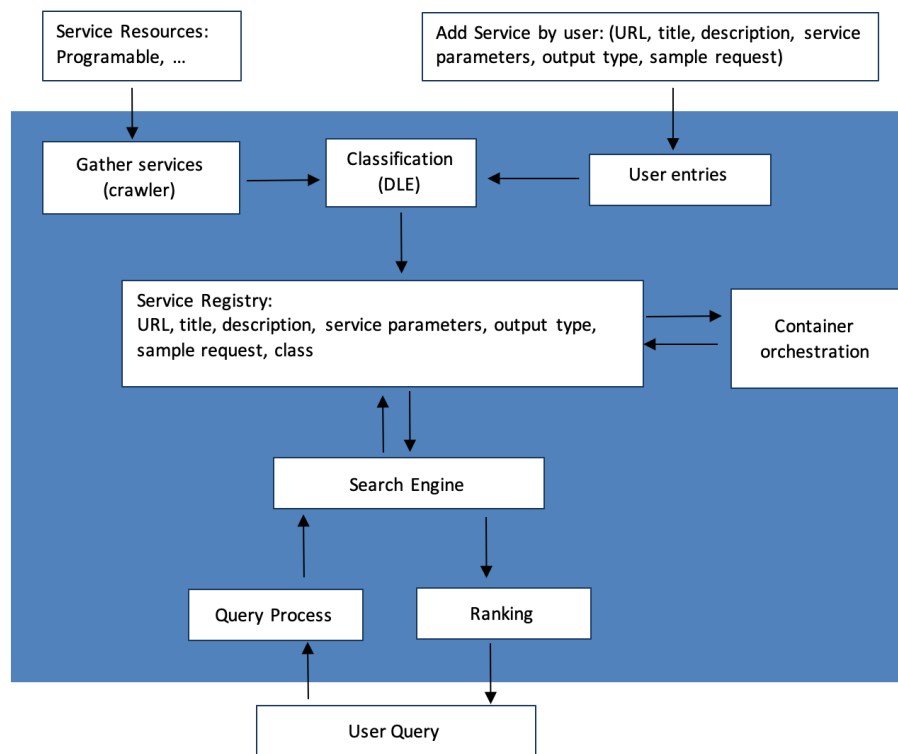


Figure 7: Local service discovery using Crawler and internal service registry

For achieving the above steps, the Model uses the following constructed component:

- **Gather services (Crawler):** Automatic extraction of services from service registries (e.g. programmable web) is done in this component. Since the services may change over time or become completely unavailable, this component needs to periodically update services data.
- **User entries:** This component is for the new services that the users created during development (manually registering). For manual registration, there are features for the service, some of which can be mandatory and some optional.

- **Classification (DLE):** This component classifies in categories the discovered services that are automatically extracted or registered by the user. In order for these services to be categorized, service categories need to be specified. A classifier which is embedded in DLE is responsible to find a category for new service/s. The input of this component can be a single service or batch of services.
- **Service Registry:** It is the place where services are stored. It can use a tabular database such as Apache Cassandra to manage large-scale data or using a simple relational database. Relational databases are great for handling complicated queries and database operations. However, an important aspect that needs to be considered for the selection of the database is its communication with a search engine. In general, a good database houses all information while a search engine can access information within that database. There are some challenges with using relational DBs and search engines. First, relational search needs to properly and thoroughly crawl a wide variety of data sources, understand the relationships between them, and provide accurate, relevant results. Second, databases can store and retrieve a lot of structured data, but search engines typically query unstructured text. As a result, it's difficult to properly design a relational database for search.
- **Search Engine:** The user's submitted and processed requests are given to the search engine and after searching in the Service Registry, the engine returns the services closest to the user's request as output.
- **Container orchestration:** The database will not only store service metadata (SQL/NoSQL), but also some container and container orchestration (e.g. Kubernetes, Mesos) may are needed.
- **Query Process:** This section is used to process the user request and send it to the search engine. Natural language processing tools or semantic structures can be used to achieve a more accurate answer.
- **Ranker:** The answers returned from the search engine in this section are evaluated and ranked based on the quality characteristics and the similarity of the service to the user's request and are returned as a list.

3 Service Classification

Service selection from among a huge pool of services can be challenging in many applications, such as service discovery and composition. As a solution, the automation of the service selection process has emerged. An automatic service selection needs to determine the category of a web service from several pre-defined categories. The web services can be annotated according to their functional and non-functional properties. However, several services can offer the same functionality. Consequently, it is a challenging task to select the right service among the ones existing in a service pool. The service selection can have the two following steps [124]:

- Services that have the same functionality are fetched from the service dataset. For this purpose, service classification is a well-known topic. Service classification tries to categorize the existing service data into a given number of classes. The objective of the service classification problem is to identify the class of a new service. In a multi-category dataset, classifiers can suggest a list of candidate categories of web services. Service classification can use semantic-based modelling and AI-based approaches.
- From the list of similar web services, a web service is to be selected. For this purpose, the quality of service parameters (availability, response time, reliability and throughput) have been used. The goal of this step is to rank the same web services based on their quality. The ranked top-k services with the same functionality can be useful for later steps such as service composition.

In this section, we focus on the first step of service selection, which is service classification. For this purpose, we conducted a case study to generate an in-depth, multi-faceted understanding of the service classification challenges in the real-world.

3.1 Related Work

3.1.1 Web Semantic Based Classification Approaches

Keyword-based search is the simplest approach, which is used in UDDI standard and the discovery mechanism (Syntactic based approaches). The idea behind this approach is that the textual description of web services is stored in the UDDI directory. The retrieval stage comprises a user or a search service, which enters a query to the UDDI directory. Semantic approaches have been used to overcome keyword search problems in search engines. There are many situations where the search is relegated to a strict keyword search. On the other hand, strict keyword searches do not allow the user to search semantically, which means information is not as discoverable. Semantic approaches use synonym words, thus leading to better results while performing text searches.

But what is the Web semantic approach? Semantic Web services are additional meaningful data to Web Services, which gave origin to the notion of Semantic Web services. In this approach “semantic web technology” and “web service technology” are integrated. Semantic technologies support a rich and standard representation of services. The main idea of the semantic web is to represent knowledge as an ontology-based language and then search it by querying it. But in some approaches, a semantic approach for automatic service classification has been used. The main semantic-based classification approach steps can include the following:

- Define and represent all the needed semantic information for a service. There are two common languages on semantic description for web services: 1) OWL-S [1] 2) WSMO [52]. Both of these languages provide a highly generic way to describe semantic information about a web service. Because of this, they are considerably complex and difficult. In some approaches researchers used simpler ontologies that fit better with their purposes, in others simple and newly developed ones were used where the ontology contains all the concepts (classes) and relations (properties) that are described in WSDL service descriptions.
- For classification, some approaches used automated methods based on the comparison of an unclassified new service with already available classified ones. This process uses the extracted semantic information created in the first phase.

Although the semantic-based approach is better than keyword base searches (UUID), there are some problems in semantic web approaches:

- There is no agreement on which language has to be used when describing the semantic information about web services. Although there are two main proposals which are OWL-S and WSMO. OWL-S adds an extra layer on top of a WSDL file in order to describe the web services (Figure 8).
- A variety of ontology technologies can lead to extra steps such as mapping when linking or using different datasets are required.
- Lack of agreement on which language has to be used when describing the semantic, lead to a few tools for automatically annotating web services. Therefore, the developers can have problems with annotating services automatically.
- Semantic-based approaches can be more efficient in a small-scale service with limited complexity.

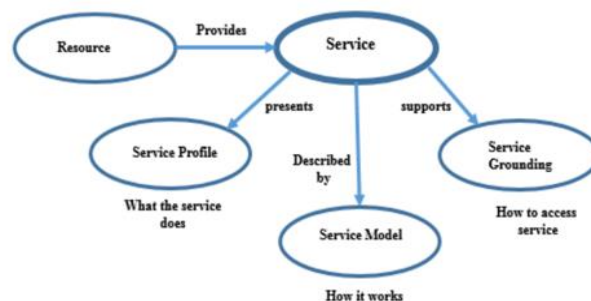


Figure 8: Web services ontology [30]

3.1.2 AI-based Approaches

Service classification tries to categorize existing service data into a given number of classes. The objective of the service classification problem is to identify the class of a new service. In a multi-category dataset, classifiers can suggest a list of candidate categories of Web services. Many researchers have investigated AI-based service classification in recent decades. This leads to the emergence of various topics in this area. In this section, related AI approaches will be reviewed.

In general, classification and semantic annotation of services are the main aspects in the service discovery, composition, and management. Some of the works have combined semantic annotations and AI approaches; however, there are two main problems with these techniques:

1. Many approaches assume that a set of annotation services is already available, but in the real-world, this assumption is incorrect and requires a lot of time and high cost for annotating services.
2. Computational complexity of these techniques is highly dependent on the number of entities (classes, properties, etc.), so it can have the problem of high-latency in huge data.

Given the mentioned reasons, AI approaches which use information extraction are better investigated in this work. Over the last decades, WSDL has more popularity; therefore, many researchers have used WSDL for feeding ML classifiers. Recently, with the emerging of REST services, researchers have focused more on service description texts. [113] have improved an existing framework by Naïve Bayesian classifier instead of Schema matching techniques. Schema matching is the problem of finding mappings between the attributes of two semantically related database schemas. [36] have introduced the AWSC model which has combined text mining and machine learning techniques. A notable point in their work is that they have considered that WSDL can include not only the NL text but also the structured data (Codes). The structured text follows Coding conventions such as naming methods, variables, etc. To this end, they have provided some simple rules such as splitting combined words. "getZipCode" will split when changing text case if it is related to Java, or " get_Quote" will split when either a space or "-" occurs. Their founding demonstrated that Rocchio algorithms provide better accuracy in comparison to Naïve Bayes. [164] have investigated SVM and ensemble learning classifiers, their result has shown that the ensemble learning classification method provided the best accuracy. During applying ensemble classifiers, there are several possibilities to achieve the highest diversity: 1) Different training datasets 2) Different training parameters 3) Different types of classifiers. Although ensemble learning classification has provided better accuracy (89%), SVM has provided acceptable accuracy by low-latency (77%). [118] have employed three fuzzy classifiers, namely, Fuzzy Nearest Neighbor, Fuzzy Rough Nearest Neighbor, and Fuzzy Rough Ownership Nearest Neighbor to classify web services. [169] have presented an approach that includes service goal clustering, and proposed hybrid service discovery by integrating keyword-based approach and an approach based on the LDA topic model.

In general, the AI-based approaches can include: 1) Traditional ML algorithms, among which SVM and Naive Bayes algorithms have shown efficient results. 2) Artificial neural networks and fuzzy logic-based approaches 3) Hybrid approaches. Most of the earlier approaches have used traditional ML algorithms; however, the performance of traditional ML learning methods highly depends on the quality of manual feature engineering. Therefore, some works have switched to neural networks and deep learning approaches which can work without feature engineering. To this end, [158] presented a deep neural network to automatically abstract low-level representation of service description to high-level features without feature engineering and then predict service classification on 50 service categories.

3.1.3 Available AI-based Approaches Datasets

The objective of existing service datasets is to serve service classification and service selection. Table 2 demonstrates the available dataset.

Table 2: Available Public Datasets

Title	Year	Description
QWS Dataset ver 1.0 ²	2007	Contains measurements of nine Quality of Service (QoS) per web service, for 365 web services and includes two additional attributes: (a) a rank of web services based on our Web Service Relevancy Function (WsRF) and (b) a class which classifies web services based on their overall performance.
QWS Dataset ver 2.0 ²	2008	Includes a set of 2,507 web services and their Quality of Web Service (QWS), the features are Response Time, Availability, Throughput, Successability, Reliability, Compliance, Latency, Documentation, Service Name, WSDL Address
OWLS-TC v3.0 collection ³	2013	More than 1000 SWS profiles which distributed to seven different domain which are: (Economy, Education, Travel, Communication, Medical, Weapon and Food)
WSRec ⁴	2014	Improving QoS Prediction Approaches for Web Service Recommendation
programmableweb ⁵	2018	includes a set of 24,046 APIs which are annotated by service name, description and link

One of the challenges in existing public datasets is the issue of obsolescence of some services that unfortunately may be out of reach. Therefore, in the rest of this section the existing datasets, their features and labels are investigated. Figure 10 demonstrates common categories which are used in literature, whereas Figure 10 the features that can be used in the service selection process. There are some important challenges for label services. The first issue is multiple categories dataset and achieve high accuracy. The second is the service annotating needs a one level categories or category/subcategories format.

- IOT Services
- Shipping Services
- Telephony Services
- Geography services
- Analysis services
- Sales services
- Human resources
- Government services
- Professional services
- Marketing services
- medical services
- simulation services
- Business services
- Communication services.
- Construction and related
- Distribution services.
- Educational services.
- Environmental services.
- Financial services.
- Food services
- Transport services.
- Cloud-based services
- Database services
- Development Tools services
- Email Services
- Engineering services.
- Health-related and social services.
- Tourism and travel-related services.
- Recreational, cultural, and sporting services.

Figure 9: Common categories/ Label for service classifiers

² <https://qwsdata.github.io/citations.html>

³ <https://github.com/wsdream/wsdream-dataset>

⁴ <http://wsdread.github.io/WSRec>

⁵ <https://www.programmableweb.com/api/>

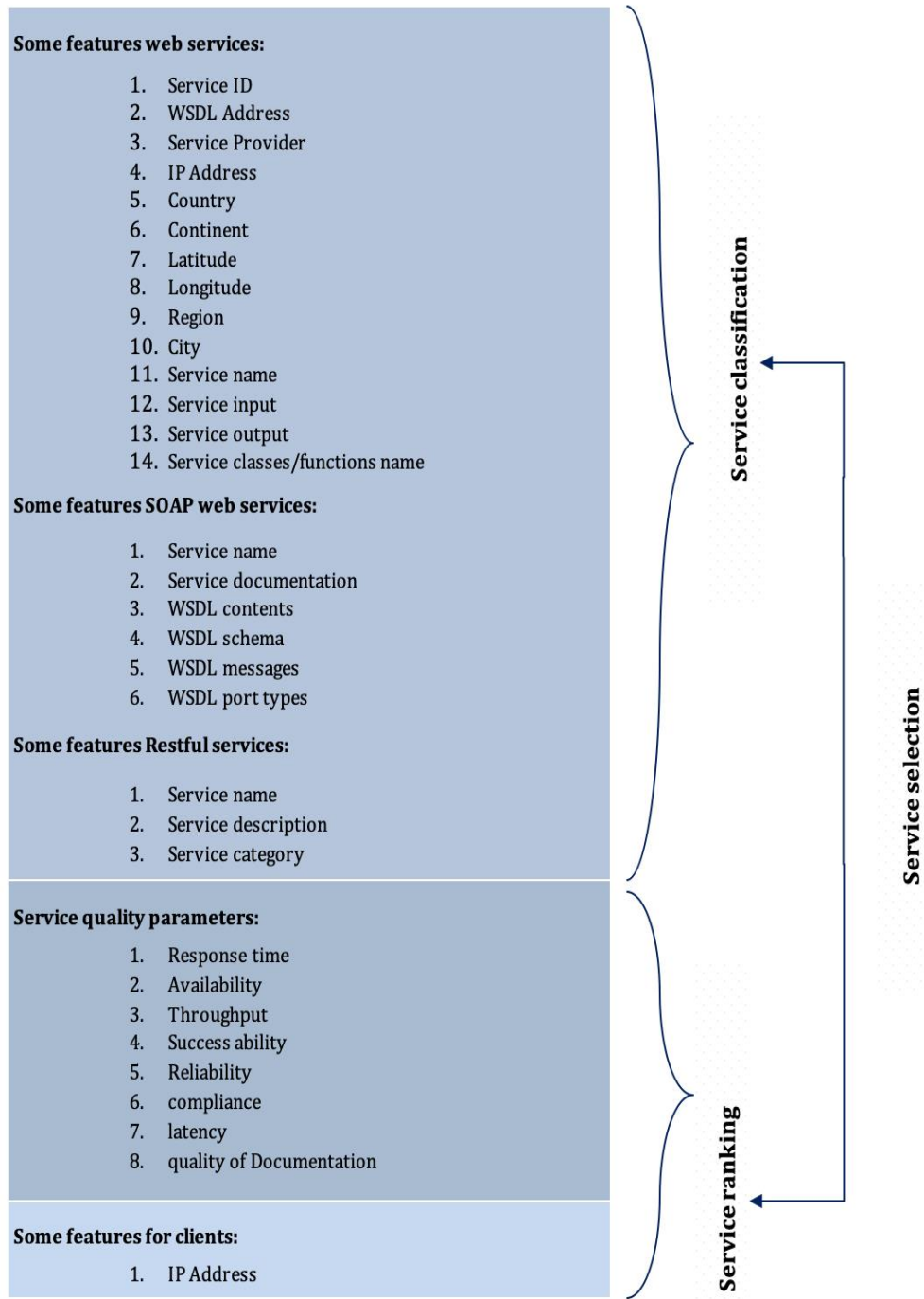


Figure 10: Service classifier possible features

3.2 AI-algorithm selection

Most semantic service classifications work well with limited well-annotated knowledge. The traditional service registry with well-defined WSDL services leads to selecting semantic modelling. However, well-annotating services is often vital because an oversized range of uncategorized data is effortlessly reachable. In fact, changing the display of service data from WSDL to REST (e.g., name, description) has also offered the choice of AI-based methods. Thus, service classification can be considered a text classification problem. As mentioned before, the early works have used traditional ML algorithms. However, more recent works tend to use Deep learning approaches. These approaches do not demand semantic or syntactic knowledge and can have better performance on the text level. Although DNN has notable advantages, difficulties like large data requirements, high-train/predict time should also be considered. This study in first phase has focused on traditional ML solutions.

While gathering collected service data the following was discovered: Most of the web service repositories have a category field to identify their service functionality. Consequently, this work has used the category field as a dataset label. The main problem is that those categories are very diverse. For example, for 6535 services we had 503 categories. There are two common approaches to reduce the number of categories:

- Map all categories to our pre-defined categories and join them manually
- Use automatic approaches to cluster categories and joining them together

Service Classification and Related AI Techniques	NLP preprocessing Steps: <ol style="list-style-type: none"> 1 - Removing punctuation 2 - Transforming to lower case 3 - Removing numbers 4 - Removing generic words of the English language 5 - Document Stemming which reduces each word to its root using stemming. 6 - Remove more frequency words in all categories which make noises
	Feature Engineering methods: <ul style="list-style-type: none"> -Bag of words (BOW) -Word Embedding
	Recategorization dataset using Clustering Algorithms: <ul style="list-style-type: none"> - K-Means Clustering - Agglomerative Hierarchical Clustering
	Classifier: <ul style="list-style-type: none"> - After data pre processing several single classification techniques such as KNN, SVM, NB, BPNN can be used. - Using ensemble classifier - Using Deep neural network, specially using Word Embeddings +CNN can provide accurate result.

Figure 11: Related AI techniques in service classification

The problem with the manual solution is that maybe two service category names are similar from a human view, but the text of the joined service description is not similar, therefore it can lead to misclassification. On the other hand, the service description text should not be ignored when joining

service categories. To solve this problem, this study has selected a hybrid model by combining unsupervised and supervised learning. Therefore, clustering approaches are used for cluster service categories base on their text. In this regard, Figure 11 demonstrates related AI techniques in service classification which will be investigate in the rest of study.

3.3 Case Study

As part of research we performed a case study to generate an in-depth, multi-faceted understanding of service classification challenges in the real-world. This process includes 7 steps:

1. Providing Dataset
2. NLP Pre-processing
3. Clustering
4. Explore clustered Data
5. Text data Classification
6. Methods for Dealing with Imbalanced Data
7. Result

Several challenges have been recognized during practical implementation which most them are well-known and have popular solutions in AI (e.g. imbalanced data).

3.3.1 Step 1: Providing Dataset

There are two important types of web service design. One is SOAP protocol (Simple Object Access Protocol) and the other being REST for Representational State Transfer. WSDL features have been used to classify web services in traditional works (see Figure 12). Features of WSDL documents are:

- Service name
- Service documentation
- WSDL contents
- WSDL schema
- WSDL messages
- WSDL port types

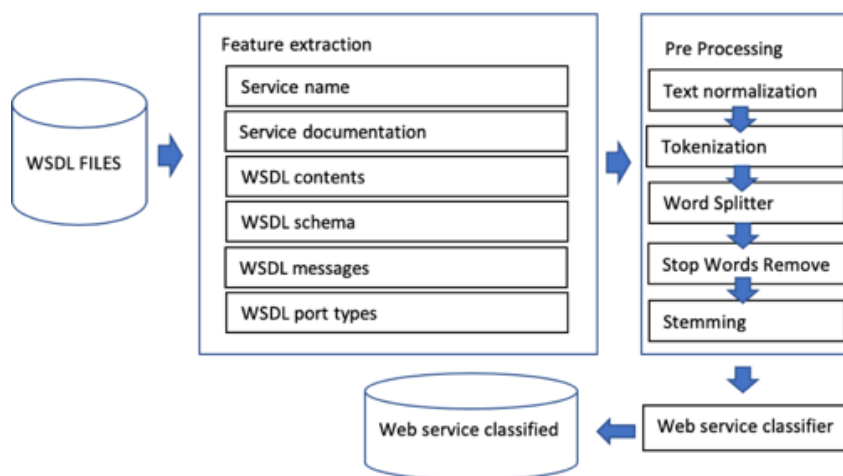


Figure 12: Proposed classification Model based on WSDL

Currently most services use the REST design. REST is, for the most part, easier to use and more flexible. It has the following advantages over SOAP:

- Smaller learning curve
- Fast (no extensive processing required)
- Closer to other web technologies
- No extra steps or tools required to interact with the web service
- Efficient (SOAP uses XML for all messages which have verbose structure, REST can use smaller message formats)

In service classification, informative features are needed, for example, service documentation in WSDL can give more information. There is an RSDL (restful service description language) concept in RESTful which is similar to WSDL. In general, WSDL is more common than RSDL. As a result, in most service repositories, the service description is just defined in one web page. Also, we tend to see that REST services are more popular in the service repositories. So, this work has provided the dataset which is independent of WSDL or RSDL. Consequently, data collection just needs service description, Service name, and Service category.

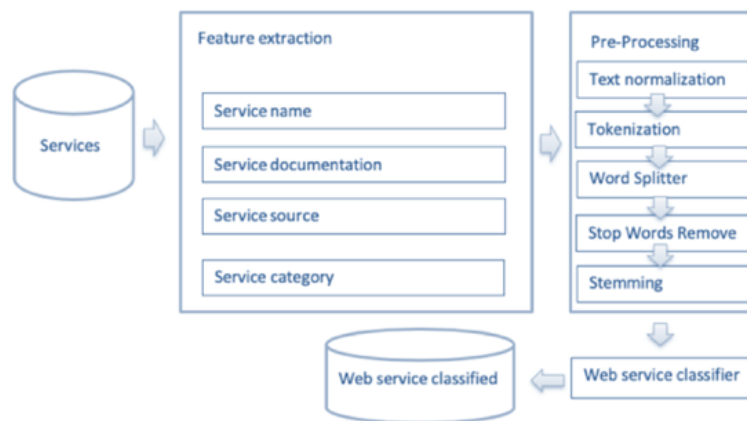


Figure 13: Proposed classification Model based on REST services

Although, some datasets are related to service classification, most of them contain information about old services that have stopped being supported. In this case study, the dataset that is provided uses 6 attributes for 6535 real web services. Those are:

- Name: The name of service
- `_id`: the random identifier for each service
- URL: It includes the URL of the service provider
- Description: Description of service functionality
- Category: The category of a Web service, from pre-defined categories
- `_type`: it shows Versions of services. REST v2.0, REST v2010.04.0, etc are example of this attribute value.
- Methods: the methods of a service class that will be provided in the next version of the data-set.
- Source: The repository source link that services are registered there.

This information is provided from "<https://www.programmableweb.com/api/>". However, this is a temporary dataset.

3.3.2 Step 2: NLP Pre-processing

The collected data is text (service name, description), therefore, as a first step, text pre-processing is required before the clustering and classification. Text pre-processing has well-known steps that utilize NLP techniques. The NLP steps being applied are:

- Step 1: The basic text pre-processing which includes process such as removing punctuations, transforming to lower case, removing numbers from the document.
- Step 2: Removing generic words of the English language such as determiners, conjunctions, and other parts
- Step 3: Document stemming which reduces each word to its root using Porter's stemming algorithm.
- Step 4: Remove more frequency words in all categories which make noises (for example, data, internet, service, etc.)

Furthermore, to decrease the number of categories, the following steps have applied:

- Only select categories that contain more than 30-rows of data (403 categories are decreased to 53)
- Use a bag of words to provide numeric data for the clustering algorithm
- Cluster data by hierarchical clustering, so the labelled classes are reduced (from 53 categories to 33)

After pre-processing, the Agglomerative Hierarchical Clustering has been used which will be discussed in the next section.

3.3.3 Step 3: Clustering

Most of the web service repositories have a category field to identify their service functionality, therefore, this study has it as dataset label. The problem is that those categories contain too much variety. For example, for 6535 services we had 503 categories, so before service classification, the number of categories needs to be reduced by clustering (sections). The objective of this section is to explore the data and to select a proper clustering algorithm. In this study, two popular clustering algorithms have implemented: 1) K-means clustering 2) Agglomerative Hierarchical Clustering

K-Means Clustering: K-means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. K-Means has the advantage on speed, being pretty fast, since it has linear complexity $O(n)$. However, K-Means has a couple of disadvantages: 1) the first is that the researcher has to select the number of groups/classes. In collected service data, the requirement is to gain some insight from the data. Therefore, specifying cluster number (k) may not help. 2) K-means also starts with a random choice of cluster centers and therefore may yield different clustering results on different runs of the algorithm.

Agglomerative Hierarchical Clustering (HAC): Hierarchical clustering algorithms fall into 2 categories, the top-down and bottom-up. 1) Top-down clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual documents are reached. 2) Bottom-up algorithms manage each document as a singleton cluster at the outset and then successively merge (or agglomerate) pairs of clusters. This process will be repeated until all clusters have been merged into a single cluster that contains all documents.

In this study, K-Means algorithm and hierarchical agglomerative clustering have both been implemented. Regarding the following reasons, hierarchical agglomerative clustering was selected: 1) Hierarchical clustering does not require specifying the number of clusters. 2) clustering process will be used to cluster collected dataset; therefore, since it will happen once, the time and a complexity of $O(n^3)$ is not an issue. HAC algorithm, also the dataset is not huge. 3) The tree diagram plot (dendrogram) of Hierarchical clustering makes more sense to understand categories that are close together.

The current study has used the following features: Service description and Service category. After the pre-processing and Agglomerative Hierarchical Clustering, the resulting classes are obtained that are demonstrated in Figure 14.

cluster_	category
0 1	Agriculture,Application Development
1 2	Payments,eCommerce
2 3	Auto
3 4	Messaging,Telephony,Tools
4 5	Mapping,Transportation
5 6	Weather
6 7	Data,Search
7 8	Real Estate,Travel
8 9	Cloud,Location,Social
9 10	Chat,Shipping
10 11	Sports
11 12	Artificial Intelligence,Entertainment,Games,Video
12 13	News Services,Stocks
13 14	Air Travel
14 15	Reference
15 16	Internet of Things,Security
16 17	Images,Photos,Science
17 18	Banking,Financial,Jobs
18 19	Music,Project Management
19 20	Enterprise,Platform-as-a-Service
20 21	Analytics,Education
21 22	Advertising,Backend
22 23	Email,Events,Marketing
23 24	Cryptocurrency,Government
24 25	Bitcoin,Business,Database

Figure 14: Agglomerative Hierarchical Clustering result for service categories

3.3.4 Step 4: Text Classification

Text classification is one of the most important tasks in Natural Language Processing (NLP). Machine learning and deep learning algorithms only take numeric inputs. The *most popular methods* for converting text to numbers are:

- **Bag of words (BOW):** A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things: A vocabulary of known words. A measure of the presence of known words.
- **Word Embedding:** is one of the most popular representations of document vocabulary. It is capable of capturing the context of a word in a document, semantic and syntactic similarity, and relation with other words.

The current study has used TF-IDF which is popular approach in NLP. To implement BOW /TF-IDF approach, the most popular python libraries NLTK (Natural Language Toolkit) And Sklearn were used.

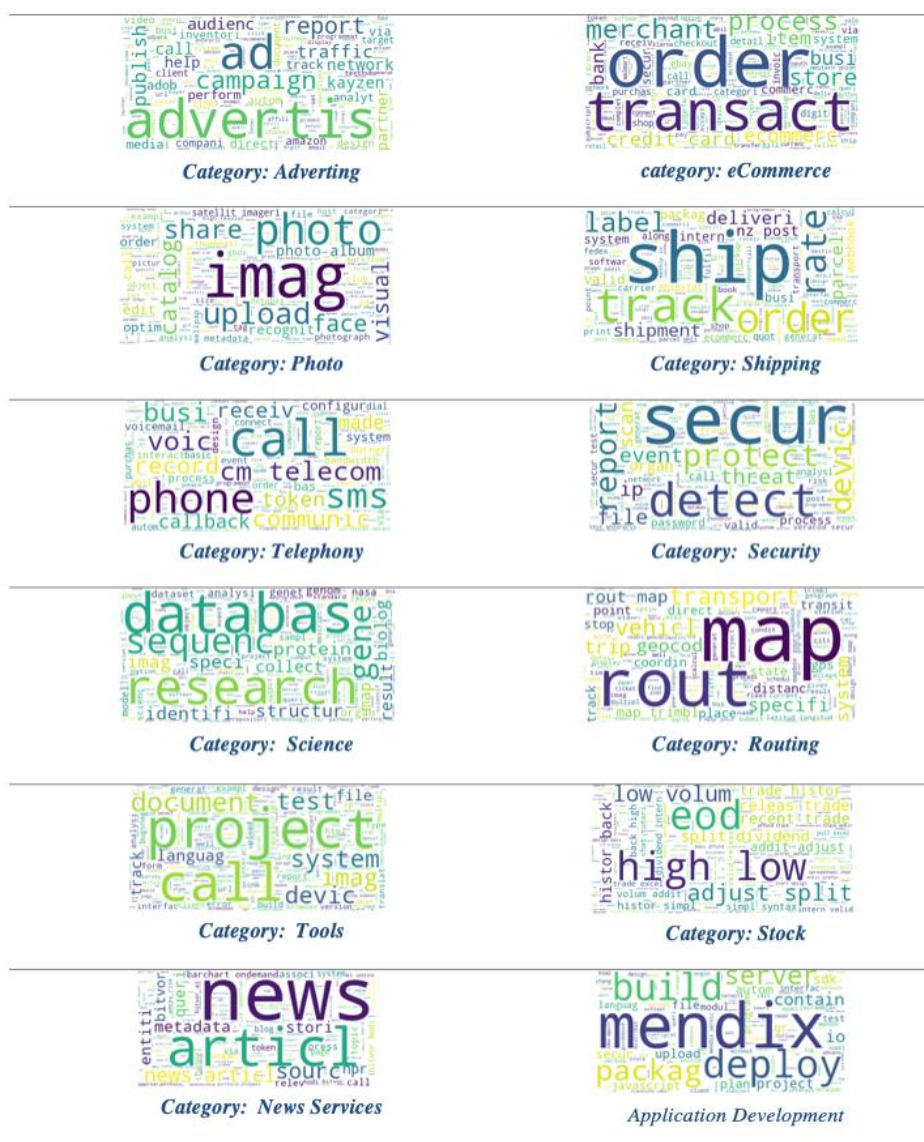


Figure 15: Word cloud

Regarding Model evaluation, accuracy was considered as reliable metric, however, the skew in the current dataset class distributions is sharp. As a result, the accuracy can become an unreliable measure of model performance. Figure 16 shows that collected data is imbalanced. Machine

learning algorithms have trouble learning when the data are imbalanced. IN imbalanced data one or more classes.

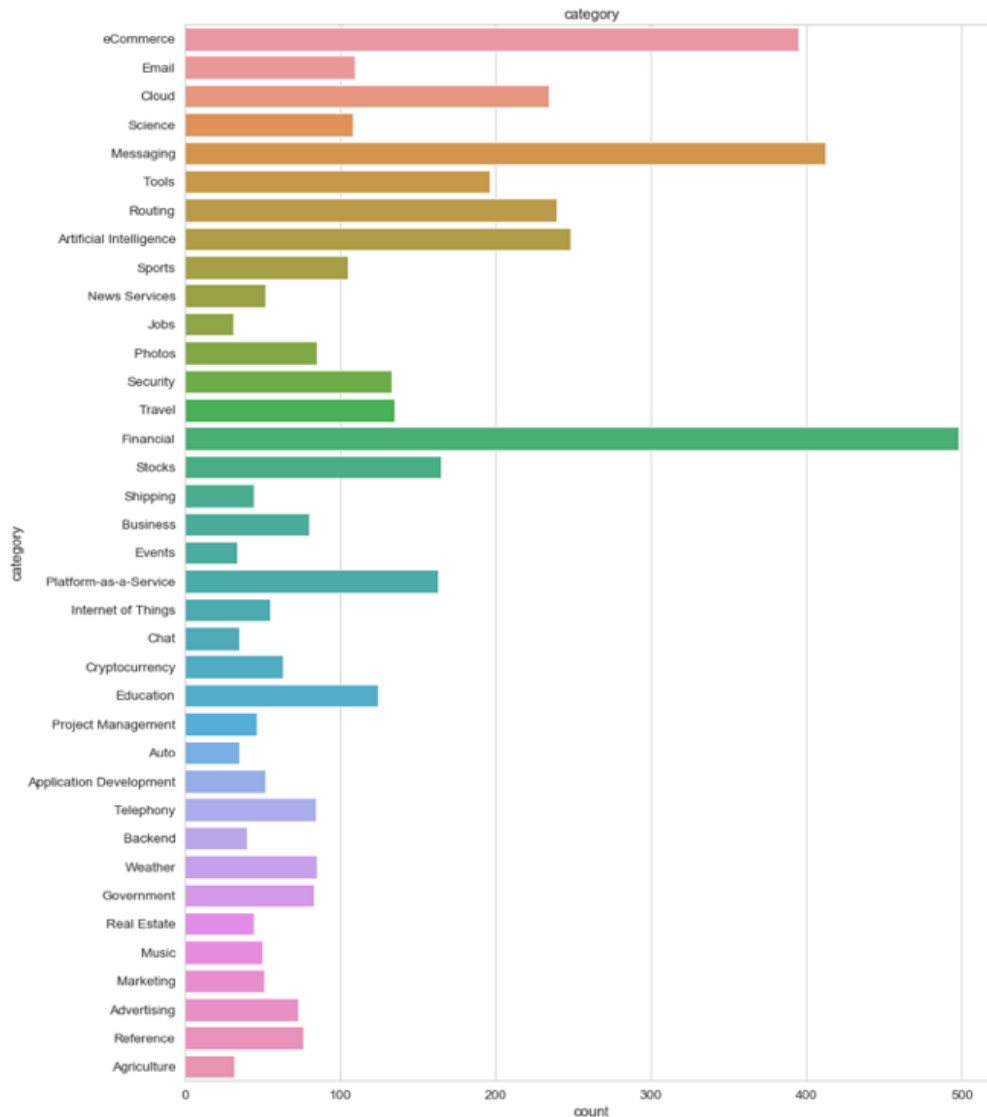


Figure 16: Bar plot for clustered data

3.3.5 Step 5: Methods for Dealing with Imbalanced Data

There are some methods for dealing with imbalanced datasets. These methods fall into three categories:

- Alter the data. This method uses popular techniques such as: Oversampling minority class, Under-sampling majority class, Generate synthetic samples (SMOTE)
- Alter the algorithm. This method can include a change of the performance metric, or a change of the algorithm,
- Alter feature engineering method,
- Word Embedding strategy

Alter the data (simple Oversampling minority): Oversampling has traditionally been used in dealing with imbalanced classes, so we will use the Imbalanced Learn library that contains its own Pipelines class

which naturally integrates with Scikit-learn. The results demonstrate that with simple oversampling the results have improved. However, these solutions are for regular ML problems that contain numeric or categorical data. For example, SMOTE is a common approach for imbalanced data in regular ML problems. SMOTE is an oversampling technique that generates synthetic samples from the minority class. In general, using SMOTE for text classification doesn't usually help, because the numerical vectors that are created from the text are very high dimensional, and eventually using SMOTE, the results are the same as if we simply replicate the exact samples to over-sample. Therefore, for a small dataset, we can apply simple Oversampling minority class or Under-sampling majority class or tuning algorithm to weigh to the classes.

Alter the algorithm (Weighted Support Vector Machines): Support vector machines are examples of discriminant functions that do not give us conditional probabilities. They are frequently used for solving problems in NLP as they handle cases of high dimensions quite well. For this purpose, the sklearn, SVM and LinearSVC library was used.

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension. Methods to generate this mapping include neural networks, dimensionality reduction [116], etc. Popular word embedding models in use today are: Word2Vec (by Google), GloVe (by Stanford) and fastText . For using these pre-trained word embedding models, there are four essential steps:

- Loading the pretrained word embedding
- Creating a tokenizer object
- Transforming text documents to sequence of tokens and pad them
- Create a mapping of token and their respective embeddings

The other benefit of word embedding is that it maps high-dimensional categorical variables to a low-dimension. To this end this approach can also help with dimension reduction. In most sources, Word Embedding is used with deep learning (Keras lib). So far, the current result shows long train/predict time.

3.4 Results

In the performed case study, we have mostly applied supervised learning (classification) algorithms on clustered data. In summary we have worked with the following algorithms and libs:

Table 3: Dataset Information-1

Problem	Data	Methods	Libs
NLP	Text	LinearSVC, Naive Bayesian, SVM, Random Forest Classifier, Deep Learning, MLPClassifier, MultinomialNB, GradientBoostingClassifier	Sklearn, Keras, Gensim(Glove), Pandas, Seaborn, Imblearn, NLTK, TfidfVectorizer

Table 4: Dataset Information-2

Records	Categories	Train/Test split	Preprocess Steps
Row Dataset	403	-	-
Clustered Dataset		80%-20%	1-Just select categories with more than 30-rows data (403 categories are reduced to 53 category) 2-cluster data by hierarchical clustering 3-to increase data and cover more categories we merged some categories with more less 30-rows.

Table 5: Case Study Result

Metric	MultinomialNB	LinearSVC	SVM	Random Forest	Gradient Boosting	CNN with GloVe
Feature engineering	BOW/TF-IDF	BOW/TF-IDF	BOW/TF-IDF	BOW/TF-IDF	BOW/TF-IDF	Word Embedding
Accuracy/imbalanced data	< 60%	< 60%	< 50%	< 50%	< 50%	-
Accuracy/balanced						
Used SMOTE and Random oversampling	75%	78%	79%	<72%	<70%	-
Training time	< 2s	< 2s	< 2s	< 5s	< 7s	< 30s

3.5 Conclusion

In this Section, we reported the findings of our research on the topic of service classification, aiming to identify important topics, approaches, and existing challenges. Besides, the collection of data and the implementation of proposed AI-model was demonstrated as a case study. In fact, service classification is a part of service selection that helps reduce search space. The main approaches in service classification fall into semantic and AI-based categories. This report is exclusively focused on AI-based approaches. Moreover, the current study was more focused on service meta-data classification. However, if service codes are also available, topics such as code mining and source code classification need to be investigated as well. These topics will be further discussed on service discovery (from code repositories) or automatic code generation section.

Regarding service data collection based on their link, the following challenges should be considered:

- During the collection of service descriptions authentication is required
- Some service URLs needed basic HTTP authentication.
- Most of the responses of old services can be in WSDL format.
- One of the challenges in existing public datasets is the issue of obsolescence of some services that unfortunately may be out of reach.

- To provide more data, researchers may use several repositories. Different repositories return distinct JSON information about services. So, if we need to join these data, the data set can have extra fields with a lot of invalid/empty values.
- One of the possible ways for collecting more data is using the existing public datasets to add more services. If researchers use public datasets, they should apply the same annotation strategy for input data; therefore, they should select the datasets which has a clear annotate process.
- After researchers collected as much data as possible, dealing with an imbalanced dataset will be an important issue. The reason is some of the service categories such as economy and education have more available services compared to others.
- Most of the web service repositories have a category field to identify their service functionality. Therefore, this research has used the category field as our dataset label. However, the main problem is those categories have too much variety. For example, for 6535 services we had 503 categories. Clustering can be a good solution in these situations.

Text mining, also known as intelligent text analysis refers to the process of extracting information and knowledge from unstructured text. Regarding feature extraction, the following challenges can arise [113]:

- Have a low grammatical quality.
- Punctuation is often ignored.
- Spelling mistakes and snippets of abbreviated text are present.
- Different services are implemented by different development teams, and these teams may use different coding conventions.
- Two services conceived for a particular task may have a syntactically different interfaces, such as `sendMail (ns:email e)` and `sendMail(xs:string from, to, subject, body)`
- Classes have different naming methods (sometimes Abbreviation and ambiguous). BCS, the different developers developed them.

Regarding supervised learning algorithm selection, there are different types of classification techniques:

- According to the available literature and the current study results SVM has shown better results in individual classifiers.
- Ensemble classifiers can provide more accuracy than single classifiers. However, researchers need to try a couple of individual traditional classifier on training & testing datasets. Also, there is a need to provide several types of ensemble systems, for example, several subsets of dataset and each subset is used for learning of one classifier. Then evaluate the results, training time, prediction time in order to provide a lightweight training model with low-latency.

In the web service selection phase, generally all the Web services have non-functional parameters. As the number of web services increases the more time and effort is required during the selection process, and as a result, it affects the performance of the system. Finally, according to literature, there are some important challenges that researchers need to consider:

- The Quality of services (QoS) parameters are conflicting and difficult to compare. These conflicts among QoS parameters lead to inter-dependency between them, which cannot be ignored during service selection.
- Service selection service should be flexible enough to incorporate the weight values provided by the end user/expert and weights calculated using a mathematical model.

4 Service Registry

This section presents a way of extracting lists of services from different sources. These sources can be service code repositories (e.g., GitHub, GitLab, and BitBucket), lists of web services such as programmable, and service registries such as Docker Hub. As can be seen in the extracted datasets, the discovered services present a great heterogeneity; For this reason, creating from scratch a service registry that allows API connection to other SmartCLIDE components, such as performing searches based on service descriptions or titles, is unfeasible due to its degree of complexity. Therefore, under these circumstances, using a high-level architecture can be beneficial in terms of cost and time savings.

We must ensure that the service registry has homogeneous fields and therefore we must consider an architecture that stores such data either by using an intermediate database or by transforming the discovery service data into a similar structure for all of them. The proposed service registry should allow quick searches based on text descriptions or titles of the stored services. A possible solution would be to store the Discovery Service data as follows: title, description, service reference. These service references could be stored in a NoSQL database which will be dependent on the Service Discovery component. Therefore, we propose the possibility of using high-level architecture for the registration of the services along with the preprocessing of the data before being indexed.

4.1 Introduction

Repositories provide a meeting point for developers to evolve the work they host. The use of repositories quickly became widely adopted by companies in the development environment, with the reason being that the whole development is stored with the record of all the changes of the project's history. These logs provide information on which developers contributed the most or roll back to previous versions to identify and fix problems that have arisen in new versions. The vast majority of software development companies, for privacy reasons, use private repositories that do not make their code available to external people, but some developers upload their project's code to public repositories such as Github, Gitlab, Bitbucket, and others. The use of public repositories encourages collaborative work that integrates developers who are interested in the project. In this way, public repository projects improve and evolve (improving their functionality, security, quality, etc.) by detecting and solving problems that the original developer has not detected. For this reason, version control systems are a tool that has to be taken into account in any software development.

Repositories have contributed greatly to software development, creating a breakthrough in the free software development community. However, there is currently a large number of projects with similar names, technologies, themes, and almost identical objectives, which complicates the task of searching for a project or service that meets the needs of the developer who is searching for one. In addition, the existence of multiple repositories means that some projects are often being replicated, in the sense of being different versions due to the contributions of different users in different ways. However, the vast majority of repositories include data and metadata that can be used by tools to try to carry out a search process automatically. In this way, this tedious task can be performed in a much simpler way in its elaboration, more accurate in its search, and much faster in its development. To this end, tools or systems must be developed that, given a token or a series of tokens, are capable of performing a search for the services that implement this functionality, using the data and metadata presented by each of the

repositories. For this reason, a tool has been created for searching and extracting repositories that correspond to a specific token or a set of tokens. Subsequently, this tool carries out a selection process of the extracted services that best adapt to the user's needs so that this service or set of services can be included in the software project in which the user wishes to integrate it.

4.2 Related work

Web-based public repositories have a standard structure in their API responses (e.g., JSON, XML). These structured responses provide automatic crawling using web-based repositories, which due to collect an extensive dataset collection. Therefore, over the past decades, the majority of research in the aspect of data collection has been emphasized on web crawling [86]. However, this form of collection creates noisy data. To this end, much of the current literature on collecting data pays particular attention to the use of Natural Language Processing (NLP) and feature extraction techniques [43],[128],[129]. According to literature gathering data can fall into two categories: “Gathering software data which include source code and metadata [77] [85] [93] [98]” and “Gathering service data which usually includes service meta data. The service can be considered as resemble software that performs a single or a few limited tasks”. The significant difference between software and service data is that the source code of service data cannot be available. In general, databases that include source codes have been utilized for smart software development tools, such as code completion [123] [146], code search [14], etc.

More recent attention [121] [122] has focused on the Github APIs, which can help to define software features. The following data can be extracted from GitHub APIs: 1) User rate 2) ReadMe file 3) Software License 4) Vulnerability alerts 5) Last commit date 6) Requirement of software 7) Number of issues 8) Data or parameter Media type 9) Number of pull request comments. Prana et al. [122] have provided a manual annotation of 4,226 README file sections from 393 randomly sampled GitHub repositories. They have used extracted data for multi-label classification. However, In real-world problems, manual annotation can cost more and suffer from human mistakes, as well. The majority of source code collections apply code mining approaches [2] [121] which assume code as structured text.

Collecting approaches that only gather service or software meta-data information can be used for automatic service discovery and service composition. Nabli et al. [107] have discussed the Cloud-based services and their discovery challenges. They have proposed a model called CSOnt which allows the crawler to automatically collect and categorize Cloud services using Latent Dirichlet Allocation (LDA). However, their services include well-annotated cloud services such as amazon services or Azure, which provide specific services. They have not provided details on the services which can provide multiple functionalities with less documentation. According to the literature, most proposed datasets either include software data (source code and metadata) or are performed for a service discovery process that only includes service metadata. This study proposes an approach that provides a large labeled dataset that includes software and service information.

4.3 Service extraction, integration, and negotiation techniques in multi-agent paradigm

This section discusses and investigates service data collection from web pages, code repositories, and classic registries. The creation of the data frames is going to be discussed below based on the information that can be obtained from each method. The main workflow is detailed in the following Figure.

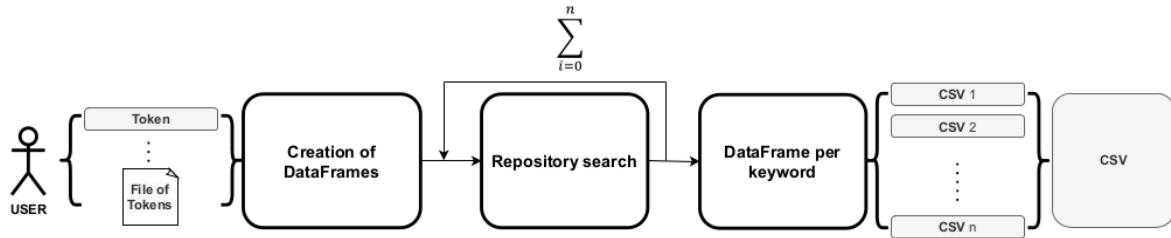


Figure 17: Pipeline of the service extraction process

4.3.1 Gathering services from websites

In general, a website offering services includes names, descriptions, and service links. The service binding contains the WSDL of the service or a list of service methods, inputs, and outputs. The main source used to gather information has been the programableweb, which has a long list of application programming interfaces (API) services, software development kits (SDKs), source code, mashups, and API-libraries. All of these are being kept up to date and classified, along with a glossary and a classification of services by keywords. Using crawling techniques has scoured the web and it has been able to create a series of scripts that obtain the data provided by the website to finally create a series of data sets which will be detailed bellow. However, only well-known services have full annotations. Based on what the web page provides, the final entries sought to build the data set have been:

– In terms of obtaining APIs:

- Name
- Description
- Categories
- Number of followers
- Endpoint
- Portal URL
- Provider
- Type of authentication
- Version
- Version status
- Type
- Architecture
- Requests formats
- Response formats
- Is official?

– Concerning the software development kit data:

- Name
- Description
- Category
- Related APIs
- Date submitted
- Provider URL
- Asset URL
- Repository URL
- Languages

– Concerning the source code data:

- Name
- Description
- Category
- Source Code URL
- Repository URL
- Languages

– Concerning the mashups data:

- Name
- Description
- Category
- Date submitted
- Related APIs
- Categories
- URL
- Company
- App Type

– Concerning the API library data:

- Related APIs
- Category
- Date Published
- Languages

- Related Frameworks
 - Architectural Style
 - Provider
 - Asset URL
 - Repository URL
 - Terms Of Service URL
 - Type
 - Docs Home URL
 - Request Formats
 - Response Formats
- Concerning the framework data:
- Name
 - Description
 - Date published
 - Languages
 - Provider
 - Asset URL
 - Repository URL
 - Terms Of Service URL

4.3.2 Gathering services from service code repositories

In this section is presented the workflow followed to obtain a series of data sets through the most popular code repositories, as well as to obtain metadata for each of them through keywords searches using crawling techniques. In order to search for services, a list of 505 keywords related to API services was obtained from the previous website. To focus the search on service repositories, 30 additional service-specific keywords were subsequently added to the website, such as WSDL, SOAP, PASS, IASS, microservice, etc. It has been decided to use the three most popular code repositories - GitHub, GitLab, and Bitbucket - to perform this type of search, which are known for hosting a large number of publicly licensed code repositories. Using their APIs, both file and keyword searches can be performed.

For all these cases a script has been developed which uses the set of keywords mentioned above. This script fetches several code repositories related to these keys, additionally, the script accesses each repository found and downloads extra useful data. Moreover, the developed script has been prepared to support both bulk and targeted keyword searches, in both cases, it uses threading to increase its efficiency, but due to API limitations, fetching the data can be delayed to hours or even days. After the raw data collection, a filtering of noisy data was carried out, in order to reduce the final number of valid repositories considerably. The data collected from each service code repositories have been:

- Data set from GitHub:
- Repository URL (the user name and repository are implied)
 - Description
 - Topics
 - Number of Stars
 - Number of Forks
- Data set from GitLab:
- Name
 - Description
 - Repository URL
 - Date Created
 - Tags
 - Number of stars
 - Number of forks
- Data set from Bitbucket API:
- Name
 - Description
 - Repository URL
 - Related Website URL
 - Date created
 - Updated on
 - Owner name
 - Owner organization

Web crawling techniques have been also used to collect information from the repositories, with the Bitcuket’s public search engine with authentication. However, this technique is limited in terms of the amount of information that can be collected from the results compared to using the API:

– Data set from Bitbucket Web:

- Name
- Description
- Repository URL
- Date updated
- Number of watchers
- Related keyword

4.3.3 Gathering Services from services registries

As a service registry, it's going to be used Docker Hub containers to collect data. For the search it has been agreed to use the docker binary filtering by the number of stars, a small parameterized script has been built that makes calls to the docker binary file and collects the response to be formatted and stored, resulting in a data set with the following entries:

– Data set from Docker Hub:

- Name
- Description
- Number of stars
- Is official?
- GitHub Repository URL
- Related keyword

4.4 Evaluation of proposed extraction methodology

The results obtained from the previous approach as well as the difficulties and possible improvements will be presented and discussed in the following subsections.

4.4.1 Data sets obtained from websites

Six data sets have been obtained from this web resource with the following entries:

Programableweb	
Resource	Valid entries
API	22.295
SDK	19.394
Source code	15.389
Mashups	6.448
API-library	1.670
Framework	555
Total	65.749

Figure 18: List of Data sets obtained from websites

4.4.2 Data sets obtained service code repositories

Overall, 535 keywords have been used to construct the data sets.

Dataset from GitHub using two API access tokens took 50 hours at a rate of about 10 keywords per hour. In total, 838.588 entries of keyword-related code repositories have been fetched.

Dataset from GitLab using one API access token took 8 hours at a rate of about 67 keywords per hour. In total, 874.623 entries of keyword-related code repositories have been fetched.

Dataset from BitBucket because of API limitations, 1.000 per hour, the repository retrieval process was delayed for 6 days, using 10 API tokens. In total, the script was able to retrieve 141.840 entries of keyword-related code repositories before the data collection process was turned off by timeout.

To increase the number of web requests from 60 to 60,000 per hour, as mentioned previously, web screening techniques were used. This new implementation took 38 hours, at a rate of about 14 keywords per hour. In total, 961.908 entries of keyword-related code repositories have been fetched.

Service code repositories	
Service C.R	Valid entries
BitBucket Web	961.908
GitLab	874.623
GitHub	812.024
BitBucket API	141.840
Total	2.790.395

Figure 19: Valid entries per repository

4.4.3 Data sets obtained from service registries

Similar to service code repositories, the list of 535 keywords mentioned in the previous paragraphs has been used. The direct use of the docker binary is limited to 100 elements per keyword but is able to filter based on popularity.

Data set from Docker Hub took 2 hours at a rate of about 260 keywords per hour. In total, 357.917 entries of keyword-related code repositories have been fetched.

Service registries	
Registry	Valid entries
Docker Hub	357.917

Figure 20: Valid entries from Docker Hub

4.4.4 Missing values

In order to get an overview of the collected entries and the amount of information from the fields described in the previous section, the distribution of missing values for each resource is presented below.

As mentioned previously, only the most popular entries have complete data available, but almost all of them have a description field which can be a starting point for using machine learning methods to classify the entries, in expectation of API-library which do not contain a description field.

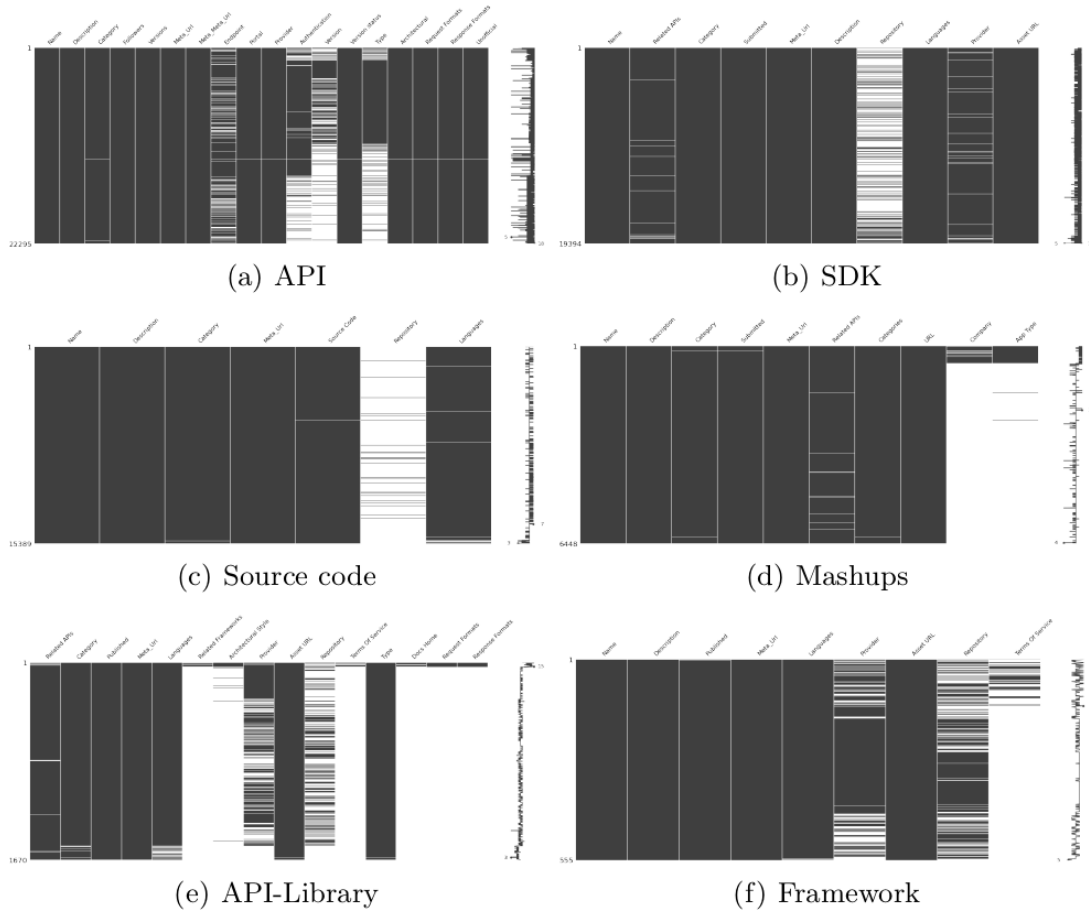


Figure 21: Missing values for the data set: programmableweb

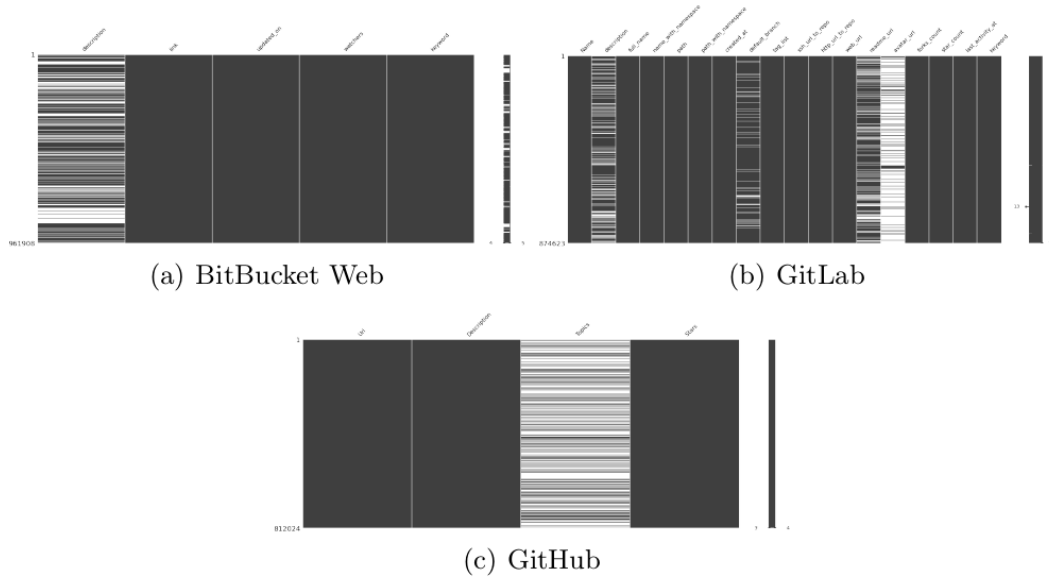


Figure 22: Missing values for the data set: Service code repositories

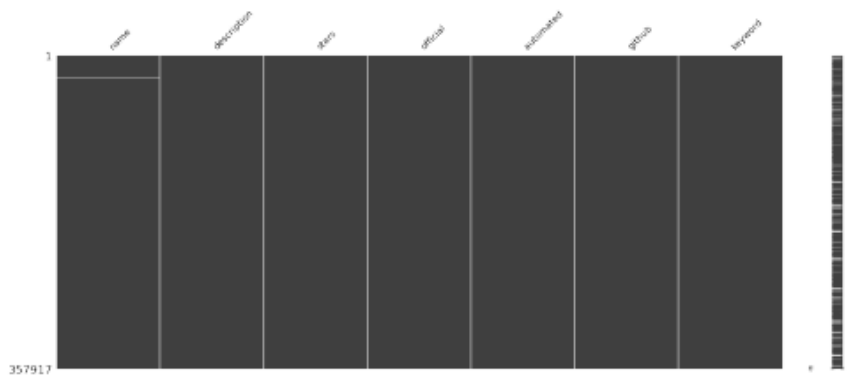


Figure 23: Missing values for data set: DockerHub

4.5 Conclusions and Future work

The process of finding and exploring repositories that fit the initial needs has become a complex task. In this section, we presented the design of a system for ingesting data from libraries, components, and repositories in a multi-agent programming environment [61]. The workflow followed for obtaining a series of datasets through the most popular code repositories was presented, as well as for obtaining the metadata of each of them through keyword searches using crawling techniques. For all cases, a series of scripts has been built that, using a set of keywords, retrieves useful data, but due to API limitations, fetching the data can take up to hours or even days. For this reason, the main source used to gather information was the programmableWeb which has a long list of application programming interface (API) services, software development kits (SDKs), source code, mashups, and API libraries that are kept up to date and classified thanks to its great community.

Combining both approaches a total of 3.214.061 code service entries have been retrieved using a total of 535 keywords, forming a data set with extra descriptions and annotations that will allow future research work on keyword-driven service recommendation using fields such as description or desired inputs or outputs from a series of petitions to an API. For future work in terms of data gathering, it has been agreed to expand these datasets to include more keywords and gather more web resources for the creation of new ones.

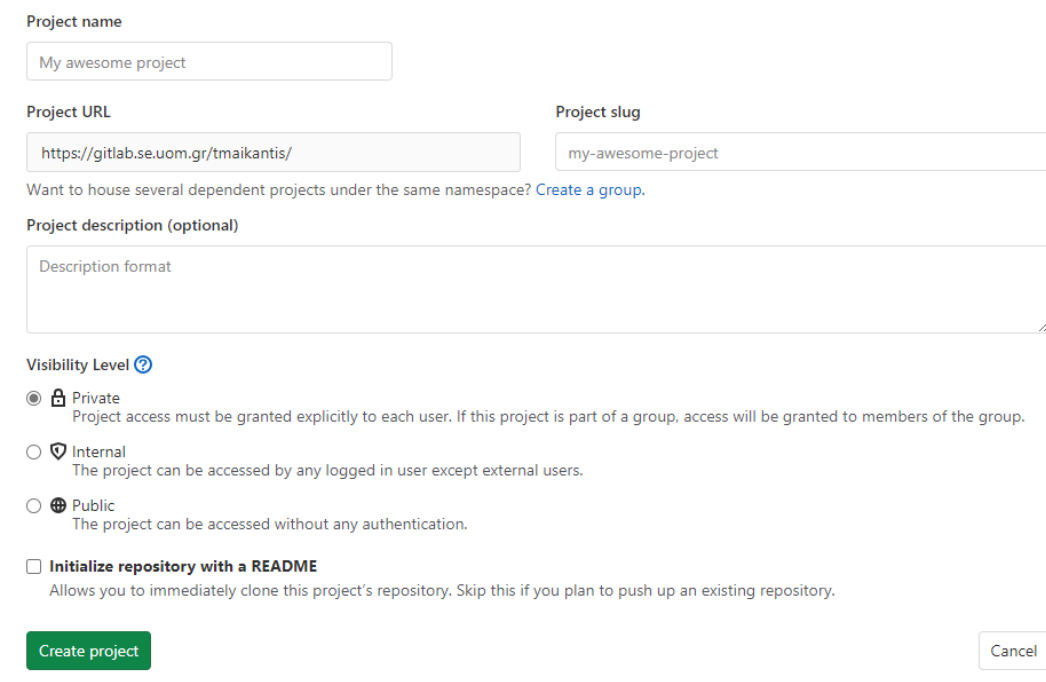
5 Service Creation and Composition

5.1 Current Service Creation Practices

In this section, we present the state-of-practice process for service creation to be used as the baseline for the development of the Service Creation component. The baseline process has been identified by exploring the used practices of the pilot providers of SmartCLIDE. Based on our analysis, to create a service, there are three necessary steps. In particular, the software engineer needs to:

- ***select a collaborative software development platform.*** As a collaborative software development platform, we have identified several solutions. The most prominent approach is the use of a cloud-based GIT version control system, such as GitHub or GitLab.
- ***develop the code per se.*** Exploring the state-of-practice in cloud-based IDEs, did not yield as many results. The most suitable and promising solution that has been identified is Eclipse Theia.
- ***employ continuous integration support.*** Similar to the first step, many alternatives have been identified. The most notable ones being: GitLab and Jenkins.

Based on the above and by considering that SmartCLIDE aims to deliver an on-premises solution, for the purpose of explaining the state-of-practice approach, we opt to use: (a) GitLab as the collaborative software development platform; (b) Eclipse Theia as the cloud-based IDE; and (c) Jenkins and GitLab as the continuous integration servers.



Project name
My awesome project

Project URL
https://gitlab.se.uom.gr/tmaikantis/

Project slug
my-awesome-project

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)
Description format

Visibility Level [?](#)

Private
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

Internal
The project can be accessed by any logged in user except external users.

Public
The project can be accessed without any authentication.

Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project Cancel

Figure 24: GitLab repository creation

Before starting the actual development process, like any other software development, the flow of creating a new service begins with analysing the anticipated functionality and identifying its key characteristics, such as: name, programming language, etc. The first development step is creating a new repository for the source code of the service (see Figure 24). In this step, we need to specify the main characteristic of our

project, such as its name, description etc. Using GitLab, creating a new project / repository is a quick and easy process. The creation of the repository is achieved through the main menu, where the software engineer is asked to provide a name, a description and a visibility level for the project.

Upon the creation of the source code repository, the software engineer needs to decide which CI tool to use. Regarding CI services, we demonstrate the support by two available tools: GitLab and Jenkins. Both tools were selected based on their overall popularity and usage, as well as the fact that they are both open source and able to be hosted on-premises. GitLab CI “comes pre-packaged” along with the Git server and offers very quick integration with the repository. On the other hand, Jenkins, which is probably the most prominent CI solution, is a second tool. The configuration of Jenkins and its pairing with a GIT repository is not so straightforward, so, next we present the process step by step. We note that Jenkins must be configured upon installation, by providing the necessary credentials and have the appropriate GitLab plugin installed. Logging into Jenkins, through the main menu (see Figure 25), we can create a new item (see Figure 26): for example a pipeline.

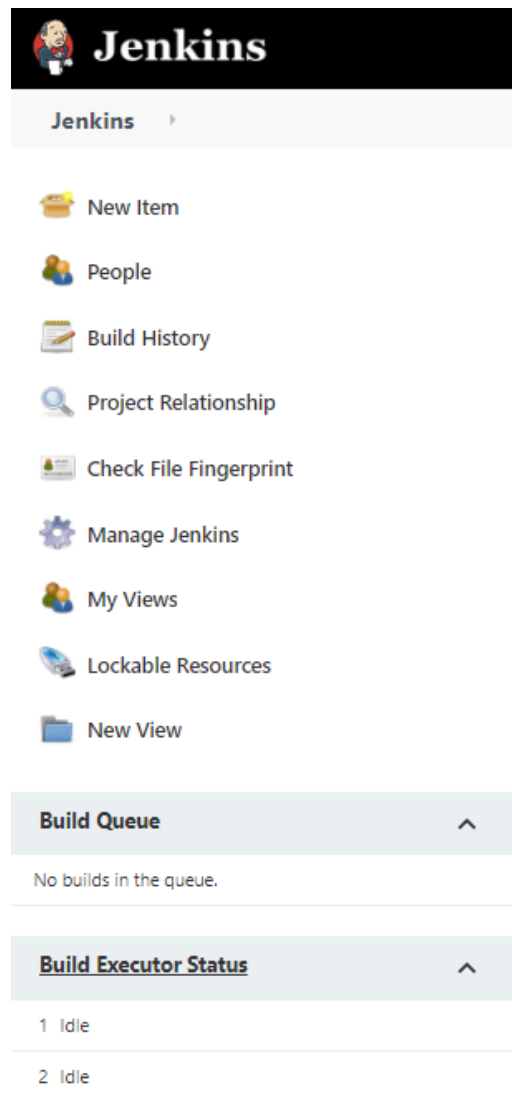









Figure 25: Jenkins main menu

Enter an item name

» This field cannot be empty, please enter a valid name

- 
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- 
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- 
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- 
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- 
GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- 
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

If you want to create a new item from other existing, you can use this option:


Copy from

OK

Figure 26: Jenkins new Item

During the creation of the pipeline, we must configure it by filling several required fields. The whole pipeline creation process is split into four main areas: *General*, *BuildTriggers*, *AdvancedProjectOptions* and *Pipeline*. Each area contains distinct functionalities about the overall Jenkins job. Starting from top to bottom, we first present the **General** area (see Figure 27). In this area, the software engineer provides an optional description for the Jenkins job and the mandatory URL of the GIT repository that is going to contain the source code.

Description

An example pipeline

[Plain text] [Preview](#)

- Discard old builds ?
- Do not allow concurrent builds
- Do not allow the pipeline to resume if the master restarts
- GitHub project**

?

Advanced...
- GitLab Connection
- Use alternative credential
- Pipeline speed/durability override ?
- Preserve stashes from completed builds ?
- This project is parameterized ?
- Throttle builds ?

Figure 27: Jenkins Job General

Next in the UI, the software engineer has access to the **Build Triggers** area (see Figure 28). In this part of the configuration, the software engineer sets up the events that are going to trigger the pipeline. For example, the selected trigger is a new code push to the repository. After selecting the event, Jenkins provides the software engineer with a web-hook URL. This URL will be used later, when the software engineer configures the web-hook of the repository, thus pairing GitLab with Jenkins.

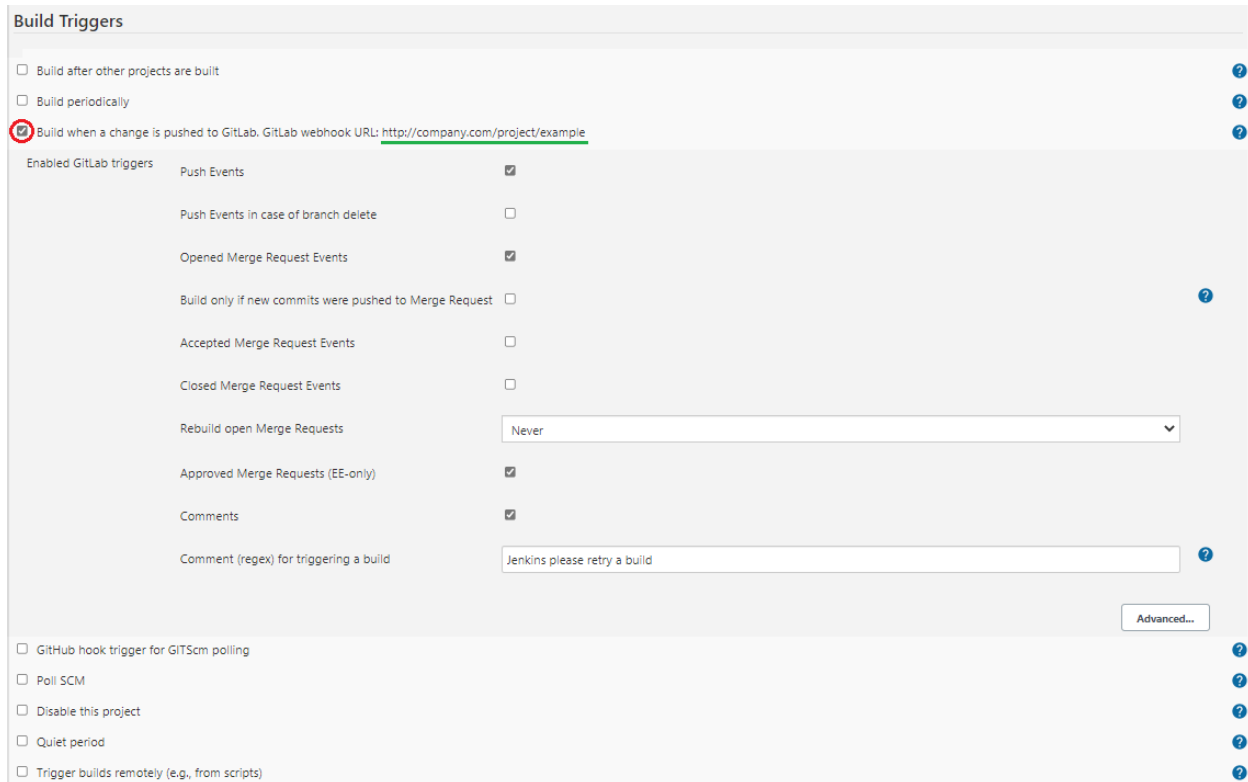


Figure 28: Jenkins Job Build Triggers

The final area corresponds to the definition of the **Pipeline** script (see Figure 29). The script is the definition of the steps that are going to be executed, when the Jenkins job is triggered. The area-1 (in Figure 29) specifies the location of the pipeline script. Jenkins provides two options: either the storage of the script inside the Jenkins job by defining it through the UI, or the retrieval of the script from Source Control Management (SCM)—this is the one selected for SmartCLIDE. By using SCM the software engineer includes the script in a project directory. By storing the script in the project repository, the developer has faster, easier and more direct access to it, in case the need arises for modifications. In area-2, the software engineer provides the URL of the project. In area-3, the software engineer specifies the set of credentials that Jenkins is going to use to get access to the GitLab server. As mentioned before, Jenkins must be configured after installation, so these credentials need to be declared prior to the creation of the Jenkins job. Lastly, in area-4, the software engineer defines the name of the file that contains the pipeline’s script. In the most common case, the pipeline steps are included inside a file called Jenkinsfile. At this point, setting up a basic Jenkins job is considered completed.

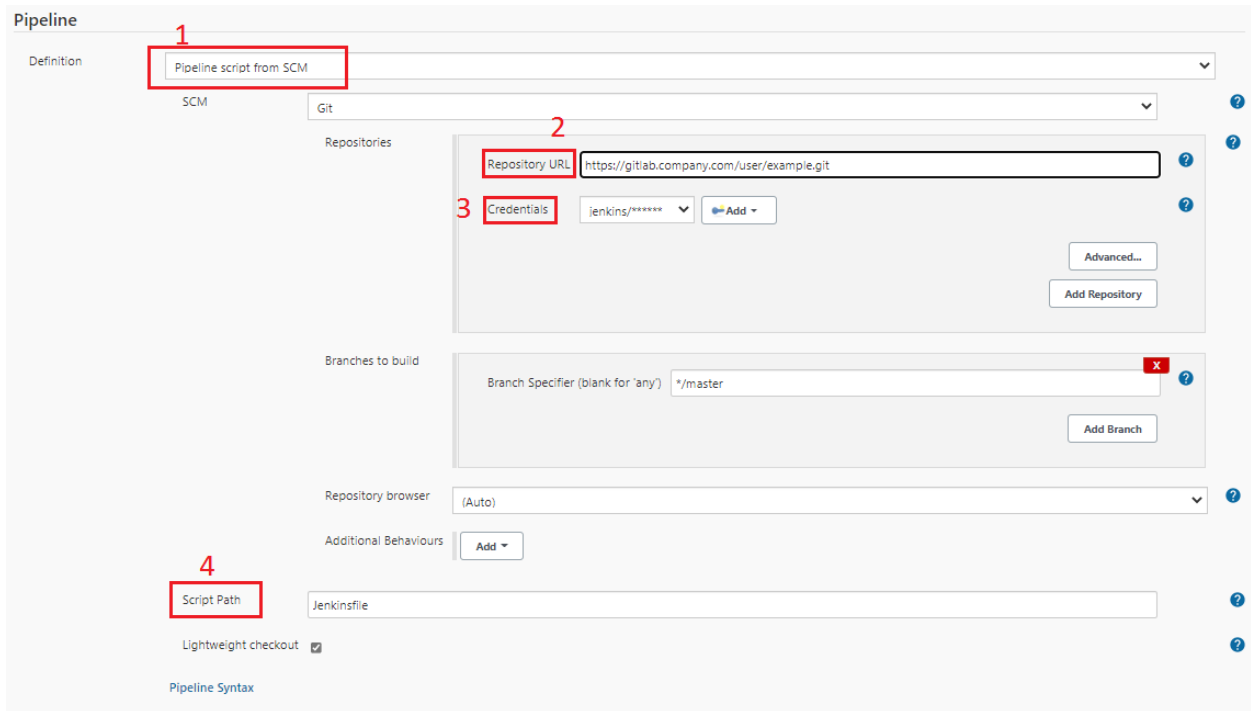


Figure 29: Jenkins Job Pipeline Script

As a final configuration step (see Figure 30), the software engineer needs to set up the GitLab repository web-hook. This serves the purpose of triggering the Jenkins job, when an event is triggered in the repository: e.g., a code push. To create a new web-hook, the software engineer must visit the *Settings* of the project and then to the web-hooks submenu.

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

Push events

This URL will be triggered by a push to the repository

Figure 30: GitLab web-hook configuration

Once the GIT repository and the Jenkins CI pipeline are configured, the software engineer can start the development process. Developing code can be achieved by cloning the repository inside the Eclipse Theia cloud-IDE, by using the built in git commands.

As it can be observed, the aforementioned process can be quite arduous and time consuming, since it requires several interconnected steps. Time spent in creating the necessary structure and / or finding problems that emerged along the way, should be instead spent for the analysis and development of the code. To this end, in Section 5.4 we present the way in which the aforementioned steps are automated in the context of SmartCLIDE.

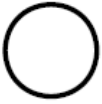
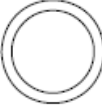


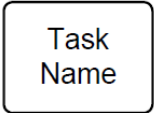
5.2 Composing Services with BPMN

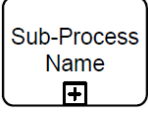
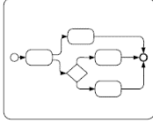










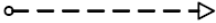
5.2.1 Basic BPMN Modeling Elements


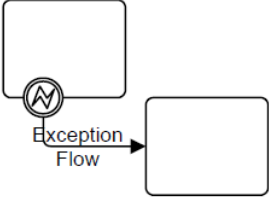
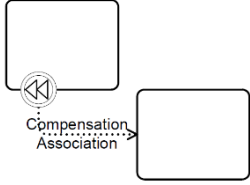
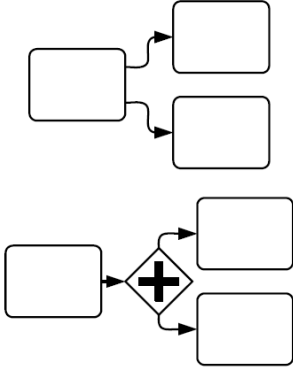
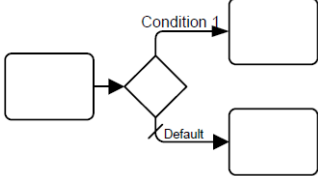
Business Process Model and Notation (BPMN) is the global standard for process modeling, maintained by the Object Management Group (OMG) institution.

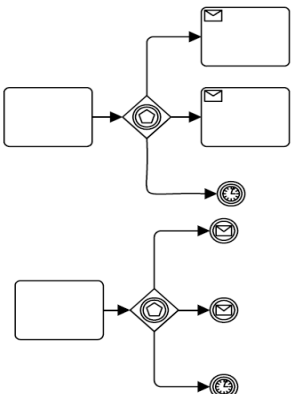
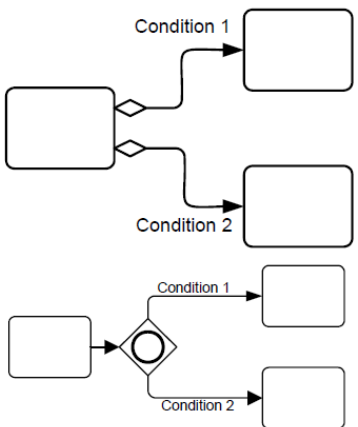
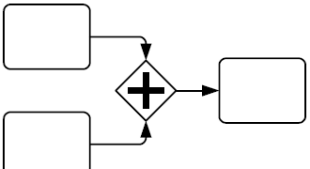
BPMN version 2.0 incorporates robust enhancements with a plethora of different notation elements that support modeling even of the most complex business scenarios. A subset of the most frequently used notations along with a short description, is presented in Table 6.

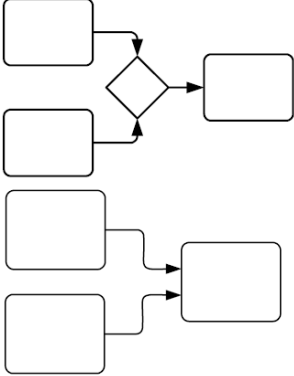


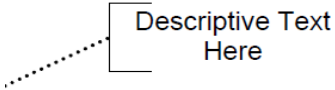
Table 6: Basic BPMN Modelling Elements

Element	Description	Notation																																																				
Event	An Event is something that “happens” during the course of a Process. These Events affect the flow of the model and usually have a cause (<i>trigger</i>) or an impact (<i>result</i>). There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.																																																					
Start	As the name implies, the Start Event indicates where a particular Process will start.																																																					
Intermediate	Intermediate Events occur between a Start Event and an End Event. They will affect the flow of the Process, but will not start or (directly) terminate the Process.																																																					
End	As the name implies, the End Event indicates where a Process will end.																																																					
More Fine-grained Event Types	The Start and some Intermediate Events have “triggers” that define the cause for the Event. There are multiple ways that these events can be triggered. End Events MAY define a “result” that is a consequence of a Sequence Flow path ending. Start Events can only react to (“catch”) a <i>trigger</i> . End Events can only create (“throw”) a <i>result</i> . Intermediate Events can catch or throw <i>triggers</i> . For the Events, <i>triggers</i> that catch, the markers are unfilled, and for triggers and results that throw, the markers are filled.	<table border="0"> <thead> <tr> <th></th> <th>“Catching”</th> <th>“Throwing”</th> <th>Non-Interrupting</th> </tr> </thead> <tbody> <tr> <td>Message</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Timer</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Error</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Escalation</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Cancel</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Compensation</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Conditional</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Link</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Signal</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Terminate</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Multiple</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Parallel Multiple</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		“Catching”	“Throwing”	Non-Interrupting	Message				Timer				Error				Escalation				Cancel				Compensation				Conditional				Link				Signal				Terminate				Multiple				Parallel Multiple			
	“Catching”	“Throwing”	Non-Interrupting																																																			
Message																																																						
Timer																																																						
Error																																																						
Escalation																																																						
Cancel																																																						
Compensation																																																						
Conditional																																																						
Link																																																						
Signal																																																						
Terminate																																																						
Multiple																																																						
Parallel Multiple																																																						
Activity	An Activity is a generic term for work that company performs in a Process. The types of Activities that are a part of a Process Model are: Sub-Process and Task, which are rounded rectangles.																																																					
Task (Atomic)	A Task is an atomic Activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process detail.																																																					

Element	Description	Notation
Process/ Sub-Process (non-atomic)	A Sub-Process is a compound Activity that is included within a Process. It is compound in that it can be broken down into a finer level of detail through a set of sub-Activities.	
Collapsed Sub-Process	The details of the Sub-Process are not visible in the Diagram. A “plus” sign in the lower-center of the shape indicates that the Activity is a Sub-Process and has a lower-level of detail.	
Expanded Sub-Process	The boundary of the Sub-Process is expanded and the details (a Process) are visible within its boundary. Note that Sequence Flows cannot cross the boundary of a Sub-Process.	
Gateway	A Gateway is used to control the divergence and convergence of Sequence Flows in a Process. Thus, it will determine branching, forking, merging, and joining of paths. Internal markers will indicate the type of behavior control.	
Gateway Control Types	<p>Icons within the diamond shape of the Gateway will indicate the type of flow control behavior. The types of control include:</p> <p>Exclusive decision and merging. Both Exclusive and Event-Based perform exclusive decisions and merging Exclusive can be shown with or without the “X” marker.</p> <p>Event-Based and Parallel Event-based gateways can start a new instance of the Process.</p> <p>Inclusive Gateway decision and merging.</p> <p>Complex Gateway -- complex conditions and situations (e.g., 3 out of 5).</p> <p>Parallel Gateway forking and joining.</p> <p>Each type of control affects both the incoming and outgoing flow.</p>	<p>Exclusive  or </p> <p>Event-Based  </p> <p>Parallel Event-Based </p> <p>Inclusive </p> <p>Complex </p> <p>Parallel </p>
Sequence Flow	A Sequence Flow is used to show the order that Activities will be performed in a Process	
Message Flow	A Message Flow is used to show the flow of Messages between two Participants that are prepared to send and receive them. In BPMN, two separate Pools in a Collaboration Diagram will represent the two Participants (e.g., PartnerEntities and/or PartnerRoles).	


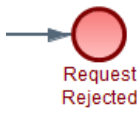
Element	Description	Notation
Default Flow	For Data-Based Exclusive Gateways or Inclusive Gateways, one type of flow is the Default condition flow. This flow will be used only if all the other outgoing conditional flow is not true at runtime. These Sequence Flows will have a diagonal slash will be added to the beginning of the connector.	
Exception Flow	Exception flow occurs outside the normal flow of the Process and is based upon an Intermediate Event attached to the boundary of an Activity that occurs during the performance of the Process.	
Compensation Association	Compensation Association occurs outside the normal flow of the Process and is based upon a Compensation Intermediate Event that is triggered through the failure of a transaction or a throw Compensation Event. The target of the Association MUST be marked as a Compensation Activity.	
Fork	<p>BPMN uses the term “fork” to refer to the dividing of a path into two or more parallel paths (also known as an AND-Split). It is a place in the Process where activities can be performed concurrently, rather than sequentially.</p> <p>There are two options:</p> <ul style="list-style-type: none"> Multiple Outgoing Sequence Flows can be used (see figure top-right). This represents “uncontrolled” flow. <p>A Parallel Gateway can be used (see figure bottom-right). This will be used rarely, usually in combination with other Gateways.</p>	
Exclusive	This Decision represents a branching point where Alternatives are based on conditional Expressions contained within the outgoing Sequence Flows. Only one of the Alternatives will be chosen.	


Element	Description	Notation
Event-Based	<p>This Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process.</p> <p>The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer.</p> <p>Only one of the Alternatives will be chosen.</p> <p>There are two options for receiving Messages:</p> <ul style="list-style-type: none"> • Tasks of Type Receive can be used (see figure top-right). <p>Intermediate Events of Type Message can be used (see figure bottom-right).</p>	
Inclusive	<p>This Decision represents a branching point where Alternatives are based on conditional Expressions contained within the outgoing Sequence Flows. In some sense it is a grouping of related independent Binary (Yes/No) Decisions.</p> <p>Since each path is independent, all combinations of the paths MAY be taken, from zero to all.</p> <p>However, it should be designed so that at least one path is taken.</p> <p>A Default Condition could be used to ensure that at least one path is taken.</p> <p>There are two versions of this type of Decision:</p> <ul style="list-style-type: none"> • The first uses a collection of conditional Sequence Flows, marked with mini diamonds (see top-right figure). <p>The second uses an Inclusive Gateway (see bottom-right picture).</p>	
Join	<p>BPMN uses the term “join” to refer to the combining of two or more parallel paths into one path (also known as an AND-Join or synchronization).</p> <p>A Parallel Gateway is used to show the joining of multiple Sequence Flows.</p>	

Element	Description	Notation
Merging	<p>BPMN uses the term “merge” to refer to the exclusive combining of two or more paths into one path (also known as an OR-Join).</p> <p>A Merging Exclusive Gateway is used to show the merging of multiple Sequence Flows (see upper figure to the right).</p> <p>If all the incoming flow is alternative, then a Gateway is not needed. That is, uncontrolled flow provides the same behavior (see lower figure to the right).</p>	
Pool	<p>A Pool is the graphical representation of a Participant in a Collaboration. A Pool MAY have internal details, in the form of the Process that will be executed. Or a Pool MAY have no internal details, i.e., it can be a “black box.”</p>	
Lane	<p>A Lane is a sub-partition within a Process, sometimes within a Pool, and will extend the entire length of the Process, either vertically or horizontally. Lanes are used to organize and categorize Activities.</p>	
Text Annotation (attached with an Association)	<p>Text Annotations are a mechanism for a modeler to provide additional text information for the reader of a BPMN Diagram</p>	

5.2.2 Events

Table 7: Events

Notation	Description
	<p>START</p> <p>Each process usually starts with a start event with a description written in passive voice. It symbolizes the event which triggered the process. A form saving, or a user action can be potential triggering events. A process can have more than one start events, but in PERSEUS context we usually stick to one.</p>
	<p>END</p> <p>Each process can have one or more end events; each one denotes the path followed to reach the termination condition. The name of the End Event should represent the state of the process. E.g. Request Accepted, Request Rejected or Terminated by User.</p>

Notation	Description
	<p>INTERMEDIATE</p> <p>Intermediate events divided in 2 categories: Catch and Throw. A “Intermediate catch” event halts the process flow until an event arrives, on the contrary “Intermediate throw” events do not pause the flow.</p>

5.2.3 Tasks

Tasks can be divided in 2 main categories:

1. **Service Tasks** are executed by the process engine and the business execution flow never pauses on them. (Except an error on a service task execution happens). Service Tasks are implemented by **Work Item Handlers**, which are classes that contain source code that is being executed by the process engine for the implementation of the task. A Process Designer can extend the [AbstractWorkItemHandler](#) and create custom implementations for the special needs of a business process.
2. **Manual Tasks** that need user interaction in order to be started and completed. The business process execution flow “pauses” on Manual Tasks and will only resume when the user notifies that the task has been completed.

5.2.4 Script Task

Script task is a subtype of a service task that usually executes a short block of code for logging or initialization purposes. We strongly suggest initiating a process flow with a special Script Task for performing central variable initialization. As Figure 31 depicts, the “Initialize Process Details” initializes variables that are critical for the overall process execution. These variables will be accessed later by many other tasks and by declaring it in a central and easily configurable place, the user enjoys less time spent in tedious repetitive tasks.

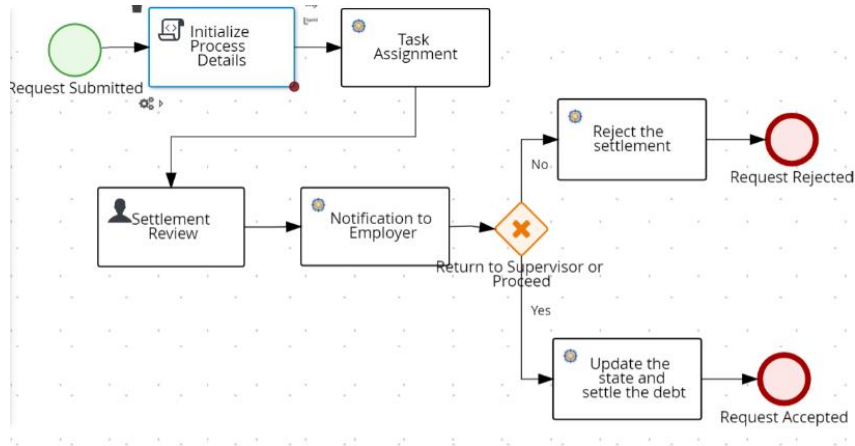


Figure 31: Example process that uses a Script Task as the first one in order to initialize some useful variables

5.3 Proposed Technological Solution for Service Composition

BPMN for service composition

Due to its nature, BPMN language offers an ideal solution for composing heterogenous services from different sources so that all together serve a common purpose and deliver business value. This can be achieved by treating created services as executors of BPMN service tasks and combine them in a structured manner by creating connections in an interactive drawing canvas.

Existing solutions

Process Automation is a rapidly growing technological field with a multitude of different implementations available that cover every possible need. From the simple automation of tedious and repetitive simple office tasks, to complex process management for enterprise infrastructure and large governmental organizations. However, in the context of SmartClide we should only consider open-source implementations for process automation, where the most mature and wide-adopted is the jBPM.

JBPM is a complete suite for business process development and execution and currently it comes in two flavors. The most mature and well-known is the complete solution framework known as jBPM Workbench where users can design, build, simulate, deploy, execute and monitor live process instances, all from a single 360 dashboard, the *Business Central*. The second one, is a novel, cloud-native implementation of jBPM called *Kogito*. While offering all modern benefits of cloud-native applications, Kogito incorporates only the headless (without UI) process execution engine and lacks a process development environment which should be deployed separately. To this extend, Eclipse Theia can play the role of the process editor if it gets enhanced with the appropriate plugins.

For the SmartClide environment we opted for the complete JBPM Business Central solution as it provides a mature, highly capable ecosystem for process design and monitoring while on the other hand, Kogito is still at an alpha phase of development and does not accomplish all of the desired features.

JBPM Business Central

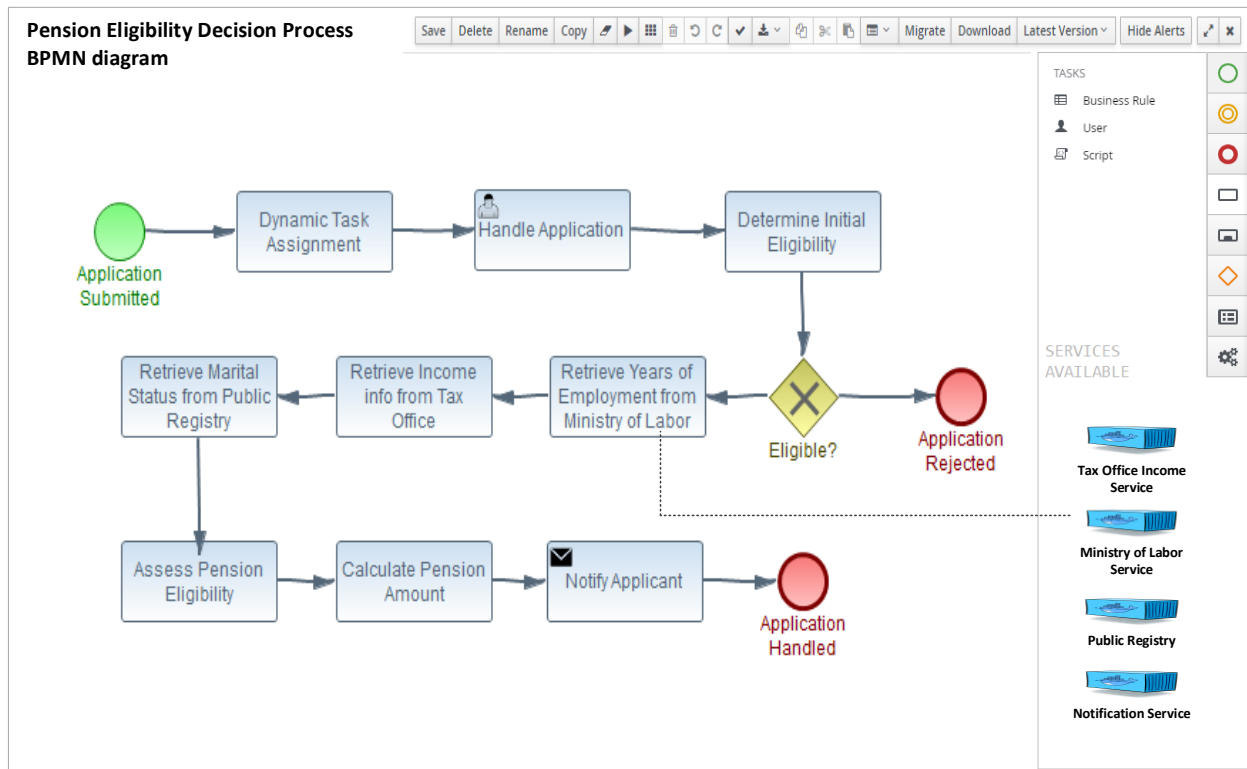


Figure 32: Example business process in the online editor of JBPM Business Central

The development of business process diagrams is performed by using the online editor which supports drag-n-drop functionality for the addition of BPMN artifacts such as User Tasks, Start/End nodes and Decision Gateways. After creating the process, the designer can easily test whether all necessary paths can be executed or whether all decision nodes contain valid check statements.

However, in the context of SmartCLIDE, in JBPM’s process editor, the developer is able to use existing services as ready, out-of-the-box artifacts, ready to be added on the current active canvas, as depicted in Figure 32. In the same figure for example, the Service Task “Retrieve Years of Employment from Ministry of Labor” is actually implemented by the external service “Ministry of Labor”, which can be seen in the lower right corner of the screen. This section hosts any service that is registered on the smartCLIDE service registry and available for being composed together with other services in a structured way that delivers business value. In this way, all internally-created services will be easily available for integration by using a simple visual development paradigm.

5.4 Proposed Technological Solution for Service Creation

In this section, we present the way in which the process presented in Section 5.1 for Service Creation has been automated, in the context of SmartCLIDE. The proposed automation is expected to reduce the effort for configuration and possible debugging. Furthermore the proposed process, would make Service Creation easier for the developer to go from the analysis of a service strait to its development, as presented in Figure 33, and outlined below.

D.2.1 SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment

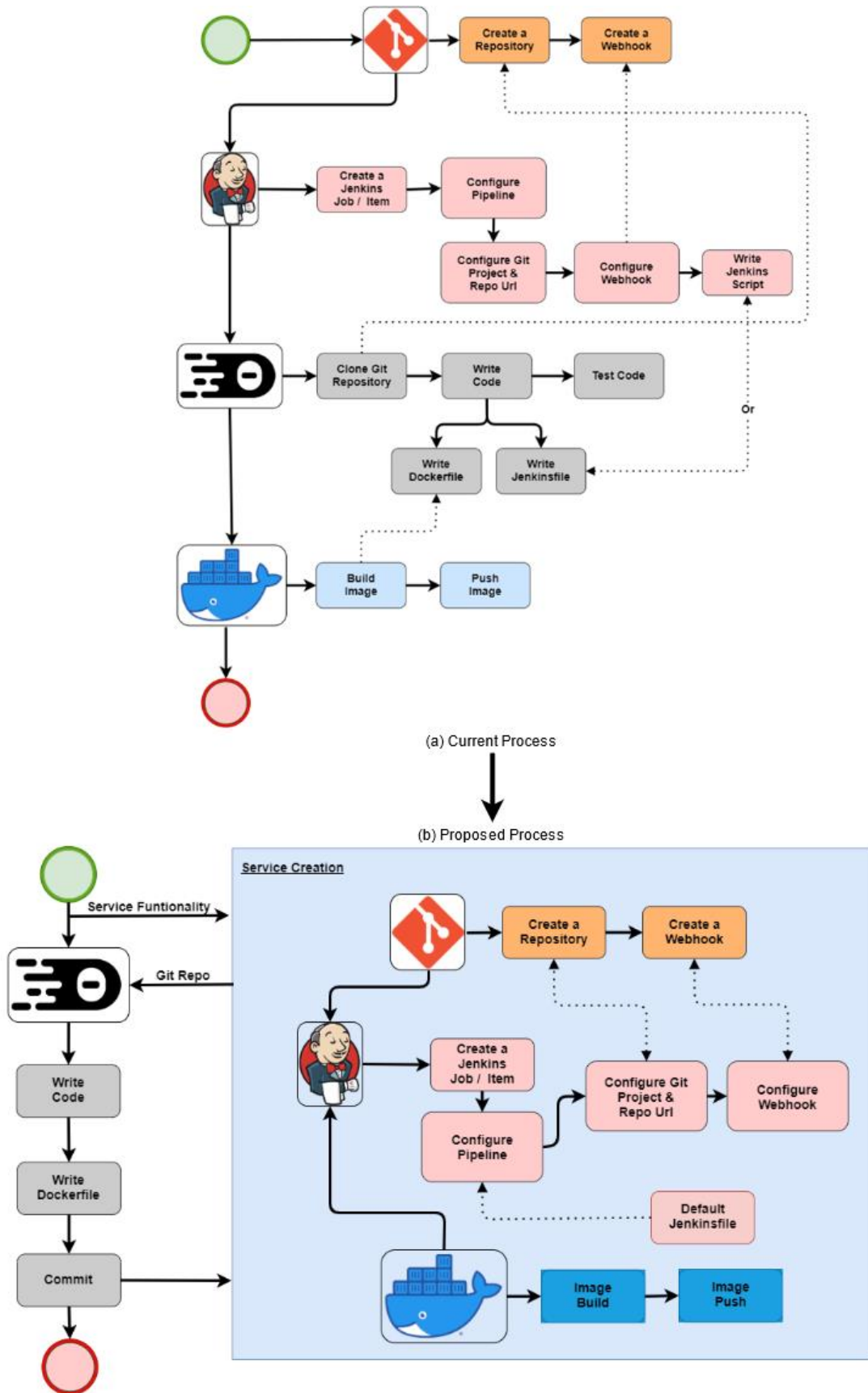
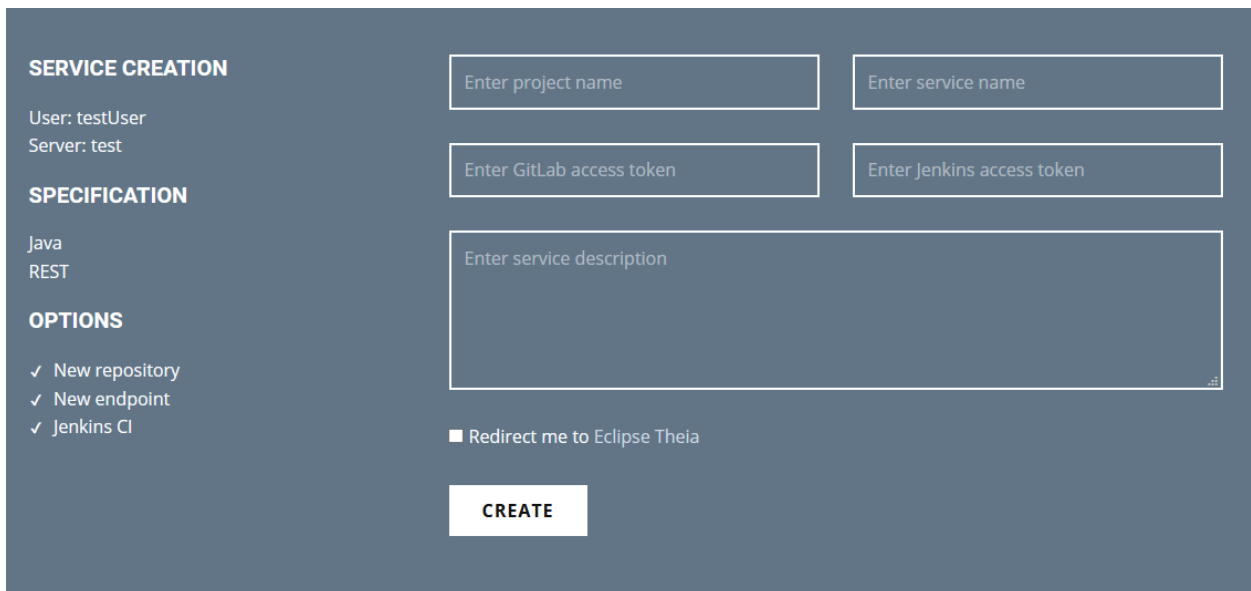


Figure 33: Process Transformation for Service Creation in SmartCLIDE

To achieve the aforementioned process, we created a REST service that accepts POST requests through multiple endpoints, and produces a desirable structure. This is achieved by leveraging the provided GitLab and Jenkins APIs, along with the information provided by the user through each request. There are several use cases for service creation, and we aim to provide support for several of them. As a general rule, there are two categories of service creation.

The first category is the creation of a completely new independent service, while the second one is the extension of an already existing project by adding a new functionality as a service (a new endpoint). The two cases are quite similar, but the first one requires some extra steps in the beginning. Based on the process described in Section 5.1, the creation of a completely new service would also require a new repository paired with an appropriate pipeline. In the first case, the software engineer will provide the required information, e.g., a project name and description, a service name and functionality and some other information, such as a GitLab token, and the repository-CI structure would be created automatically (see an example screen in Figure 34). We note, that in the final version of the technological solution, the service specification will fully comply with the SmartCLIDE schema presented in Section 3.



SERVICE CREATION

User: testUser
Server: test

SPECIFICATION

Java
REST

OPTIONS

- New repository
- New endpoint
- Jenkins CI

Redirect me to Eclipse Theia

CREATE

Figure 34: Example landing page

After the creation of the repository-CI structure, the two cases follow the same procedure. The user is redirected to the cloud-IDE (i.e., Eclipse Theia), in which the repository is automatically cloned. Then a new source file is automatically created to serve as the starting point of the new service. The source file contains skeleton code with the appropriate annotations that distinguish the service's endpoints. The generation of the source file, is based on the information that the user provided in the initial service creation landing page. The provided information includes the programming language (i.e. Java or Python), the service name, the service functionality and any other information required by the SmartCLIDE Smart Assistant (see Sections 7 and 8).

By comparing the two processes, we believe that the service creation process has been significantly speeded-up and simplified, by eliminating the need for the previous highly involved process where the users needed to use and manually configure several tools. Nevertheless, the proposed process for Service Creation and Composition will be evaluated within the pilot case providers, in D.2.2.

6 Service Specification

A specification can provide the characteristics of a software product or service. To this end, service specification is the process of providing guidelines to identify the features, requirements, and objectives of services, which can fall into two categories: functional and non-functional specification. Most web service specifications can include a description of the interface scheme (e.g., SOAP or REST), service data schema (e.g., XML, JSON), or service operations with their data payload [167].

Several specifications and standards are available in the Service-oriented Architecture (SOA). These specifications and standards can be related to reliable messaging, notification, transactions, resources, and management. Current work aims at investigating functional service specifications. A functional specification is responsible for defining what outside agents (e.g., users or APIs using a service) might require when interacting with the services. For example, services can be specified using the service functions list, their input and output parameters. On the other hand, service specification helps users select a service based on their requirements and criteria. At first, users need to describe what the service might do. Afterward, service specification can provide help to the end-user by checking if a candidate service matches some criteria or not. Web services could be any software, application, or cloud technology that uses standardized protocols to communicate and transfer data messaging. In general, there are two methods to implement web services: SOAP/WSDL and Restful services. The service concept plays an essential role in achieving proper service functional specifications. To this end, web service structure, data flows, standards have been investigated in this report.

In summary, current work categorizes services based on the following items: (1) service basic, (2) service deployment, (3) service operation, (4) service accessibility, and (5) service financial information. The findings show that service implementation evolution has a significant impact on service specification. Modern services utilize API architecture instead of soap services that have been popular in the past. Although the documentation of restful APIs describes the functionality completely, they are not as machine-readable as SOAP services. Therefore, the accurate service specification still needs human supervision.

6.1 Background

Services perform a single or a few specialized tasks, while applications contain a wide range of operations. However, the service can be considered as a resemble of a software application. Therefore, it can be expected that the integration layers will be similar in both. Web services stack can define an architecture of service concepts and protocols. These protocols and standards help services to be self-describing. Figure 35 shows not only the "web service stack" but also demonstrates popular service standards. These standards include first-generation implementation technology of services (WSDL-Soap services) and today's services (e.g., APIs, Restful). Soap services contain several standards which are proposed by industrial software vendors, while REST architecture is more close to current web technologies [18] [125].

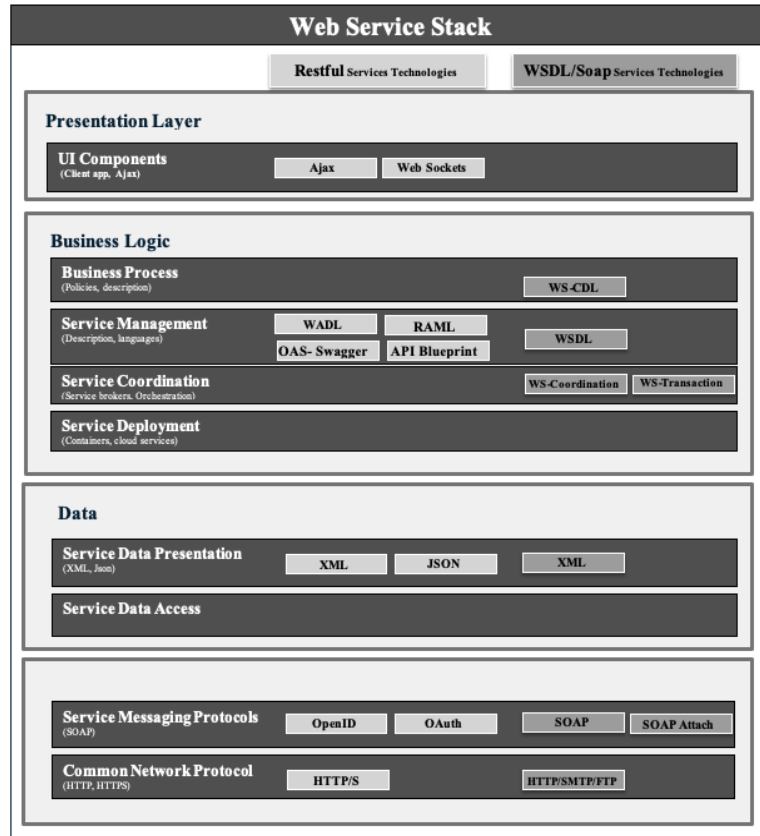


Figure 35: Web Service Stack in Restful and Soap Services [18][155]

6.2 Related Work

The available functional service specification includes related information that is needed to interact with services. These interactions typically involve the service client interaction (both Human and application). In soap services, SOAP and WSDL specify service descriptions and communication formats between services. However, REST services contain a set of generic web service design principles and guidelines. REST is an architecture that does not contain official specifications or standards. However, well-designed Rest-APIs can help clients to understand service functionality better. Based on service definition, REST services should be self-describing and self-documenting as much as possible. The Restful service principles (e.g., resources, addressability, uniform HTTP actions) can give humans a straightforward guideline for designing the Web API [155]. The main problem is Restful documentations describe APIs functionality in free text, and automated systems still wanted more machine-readable structure.

Since the growth of the service and API marketplace, the learning approaches have got the attention of researchers. In general, JSON is common interaction and data representation technology that is being used in the Rest architecture. To this end, some research topics such as Open API Specification (OAS)/Swagger [72] have emerged, which is a structured-based and machine-processable description of REST services. According to literature gathering data can fall into two categories: 1) Gathering software data which includes source code and metadata [73] [145] 2) Gathering service data which usually includes service metadata [167]. The service can be considered as resemble of software that performs a single or a few limited tasks. The significant difference between software and service data is that the source code of

services cannot be available. Table 8 demonstrates possible Functional and Non-Functional Service Features which were collected from the literature.

Table 8: Review of Functional and Non-Functional Service Features

Service Specification	Domain/attributes	
Functional	Service Description	Service Name Service Type: Rest or SOAP services Service Operations List Service keywords Service dependency Requirements License
	Important Features /Rest APIs	End Point HTTP Method (e.g.POST, GET, PUT) Operation List + Required parameters /responses for each operation
	Important Features /WSDL services	Service URL WSDL Address
	Service Data Model	Data representation (e.g. JSON, XML,) Data storage technologies
	Service Interaction	Message exchange pattern (SOAP/HTTP) Service Request API Order (e.g.Transactions step) Service Input/output for each function Relation of the service to other services
	Service Access	Service Address Service Repository Access Service Ports Service Accountability Service request limitation
	Service Deployment	Service Capsule Ability State build Version
Non-Functional	Service Performance	Processing time Response time
	Service Reliability	Number of downloads Followers Stars Last update Number of Issues, commits Downtimes for maintenance
	Service Security	Encryption Service Accountability

Service Specification	Domain/attributes	
	Financial details	Accounting method (e.g. Open source, Freemium) Price of the IT Service for the client

SmartIDE users can discover web services from internal registries and third-party repositories. Therefore, it is important to consider extractable information from third-party repositories (web, code repositories) to specify services. In fact, fully automatic annotations for discovering services can be challenging. This process is challenging for Restful Web services whose documents have been provided in arbitrary HTML pages or text format. Although there are learning approaches such as Open API Specification (OAS), human supervision can also be helpful. Moreover, several text/code mining approaches can provide more Web service features.

A popular solution can be crawling Web-based public repositories, which include a list of Web services. These resources have a standard structure in their interface (e.g., table, list) or API responses (e.g., JSON, XML). These structures provide automatic crawling using web-based repositories, which due to collect an extensive dataset collection. Therefore, over the past decades, the majority of the research in collecting data has been emphasized in web crawling. However, this form of collection creates noisy data. To this end, much of the current literature on collecting data pays particular attention to the use of Natural Language Processing (NLP) and extraction techniques. Figure 36 shows the minimum extractable features which are typical on most popular sources. The related code which is responsible for crawling can be found on Github⁶.

Gitlab	dockerhub	Bitbucket	LIB_programmablew	Programmableweb	Common Attribute
full_name, name_with_namespace	name	full_name	SDK Name	Framework Name	Service Name
description	description	description	Description	description	Description
path path_with_namespace default_branch web_url	-	link	URL	Address URL	Service URL
ssh_url_to_repo http_url_to_repo web_url	github	link_github	Repository	Repository	Repository URL
language	-	language	language	language	Language
tag_list, keyword	keyword	type keyword	Category	Category	Class/Category/ Label
avatar_url forks_count star_count last_activity_at	stars		Related APIs	Published Meta_Url	None-functional Attribute (Score)

Figure 36: Extractable features from most popular online sources

⁶ <https://github.com/eclipse-researchlabs/smartclide-task-service-discovery>

6.3 Specify services from Web service registries

In general, Web service registries include names, descriptions, and service links. SOAP services provide service functionality, messages, operations, ports in a WSDL document in XML format. However, REST services typically provide API documentation guide in HTML pages, XML documents, API guides (in XML or JSON format), etc. Also, REST services using the standard HTTP operations have a uniform interface. The popular HTTP methods for REST services are: PUT, POST, GET, DELETE. To this end, the functional service specification can follow a different strategy for the specification of REST or SOAP services. Table 9 demonstrates possible service specifications. In these specifications, we assume services as a class that contains several functions or APIs. Programable and Rapid API are well-known Web service registries. In the available information of the web registries, the GitHub URLs can be included, and the second-level web scraping script can add more information to our dataset. Extracting knowledge from code and Readme files will be discussed in subsequent sections.

6.4 Specify Services from Code Repositories

More recent attention has focused on the GitHub API, which can help to define software features. The following features can be extracted from GitHub API: 1) User rate 2) ReadMe file 3) Software License 4) vulnerability alerts 5) Last commit date 6) Requirement of software 7) Number of issues 8) Data or parameter Media type 9) Number of pull request comments. Prana et al. [122] have provided a manual annotation of 4,226 README file sections from 393 randomly sampled GitHub repositories. They have used extracted data for multi-label classification. However, in real-world problems manual annotating can cost more and suffer from human mistakes, as well. The majority of source code collections apply code mining which assumes code as structured text.

Table 9: Possible types of service specification

Attribute	Type	Web service Registry/ Extraction method	Code mining using GitHub
Service Name	Free text	Using Service name/Automatic	-
Description	Free text	Using Service Desc/Automatic	Using Readme files, Desc/Automatic Extraction
Keywords/tag	Free text	Using Service Desc/Automatic	-
Type	Checkbox	WSDL service or Restful service/Manual	WSDL service or Restful service/Manual
Service URL	URL	Using Service URL/Automatic	Using Code repo URL/ Manual
Service WSDL or doc link	URL	Using Service Desc/ Automatic	Using Service Desc/ Manual
Git URL	URL	Using Service Desc/ Manual	Using Git URL/ Automatic
Last update	Date	-	Using last commit date/ Automatic
Licence	text	-	Using License section in readme file file / Automatic
Technical requirements	text	Using Service Desc/ Manual	Using requirements section in readme file file / Automatic

Attribute	Type	Web service Registry/ Extraction method	Code mining using GitHub
Languages		Using Service Desc/ Manual	Using file extensions in git repository file / Automatic
Followers/Rank		Using Service Followers, downloads/ Automatic	Using rate option in git repository file / Automatic
Provider		Using Service Desc/ Manual	-
Output format	Checkbox (e.g, JSON/XML)	Using Service Desc/ Manual	Using code mining/ Semi - automatic
Operation list	Function <input>	Using Service Desc/ Semi - automatic	Using code mining/ Semi - automatic
Parameter	Input/Output Data Type	Using Service Desc/ Manual	Using code mining/Manual
Version	text	Using Service Desc/ Automatic	Using License section in readme file / Automatic
Capsule Ability	Bool	Using Service Desc/ Manual	Using Readme files, Desc / Automatic
Accounting method	Text	Using Service Desc/ Manual	Using Service Desc/ Manual

6.5 Functional Service Specification Schema

This service functional specification schema provides possible extractable attributes information from the web service registry, source code repositories. This information extraction can be fully automated or semi-automated. However, manual annotation during service containerization can offer more information. The schema has the following sections:

- **The Basic Service Information Item:** The basic information includes extractable features typical of most popular sources.
- **The Deployment Service Information Item:** Services are self-contained software that can be called via a network. The features like capsule ability allow the instantiation of all requirements and related APIs once at service startup as capsule services. Recently, most of the popular services have provided a prepared container for their service. This option let customer deploy a service on their local server.
- **The Function/API of Service Information Item:** Some concepts such as RMI (Remote Method Invocation)/SOAP have been provided easy management for function/operation execution. However, these concepts are mostly replaced by REST calls. The concept of Rest is to provide resources, which mostly have a specific URI (Uniform Resource Identifiers). Some Web service registries like RapidAPI⁷ service URI and operation are clearly defined in some web registries, and it is possible to extract functions and information automatically. Nevertheless, in most web registry API operations are in free text and can be extracted manually.
- **The Access Service Information Item:** There could be some prerequisites while sending requests to a Server, such as user authentication, APIs, Access Tokens, exposed ports for services, etc. This information needs to be collected manually.

⁷ <https://rapidapi.com/>

- The Service Reliability Information Item:** There are some non-functional attributes for each service web service registry or code repository to demonstrate Quality of services (QoS), such as number of service downloads, rate, developer following, number of issues in GitHub, last commit, etc. This information can almost be retrieved automatically.
- The Service Financial Information Item:** Most providers use the freemium offer for their services, which provide free basic options and paid premium versions. This information needs to collect manually.

Figure 37 demonstrates the proposed schema for Web service specification. In this schema, every element denotes an information item, and every information item has attributes that fall into two categories: 1) Required 2) Optional. Moreover, each element has a specified data type, which can be in a nested format that represents by JSON. The extraction method is the result of the information obtained from Table 10.

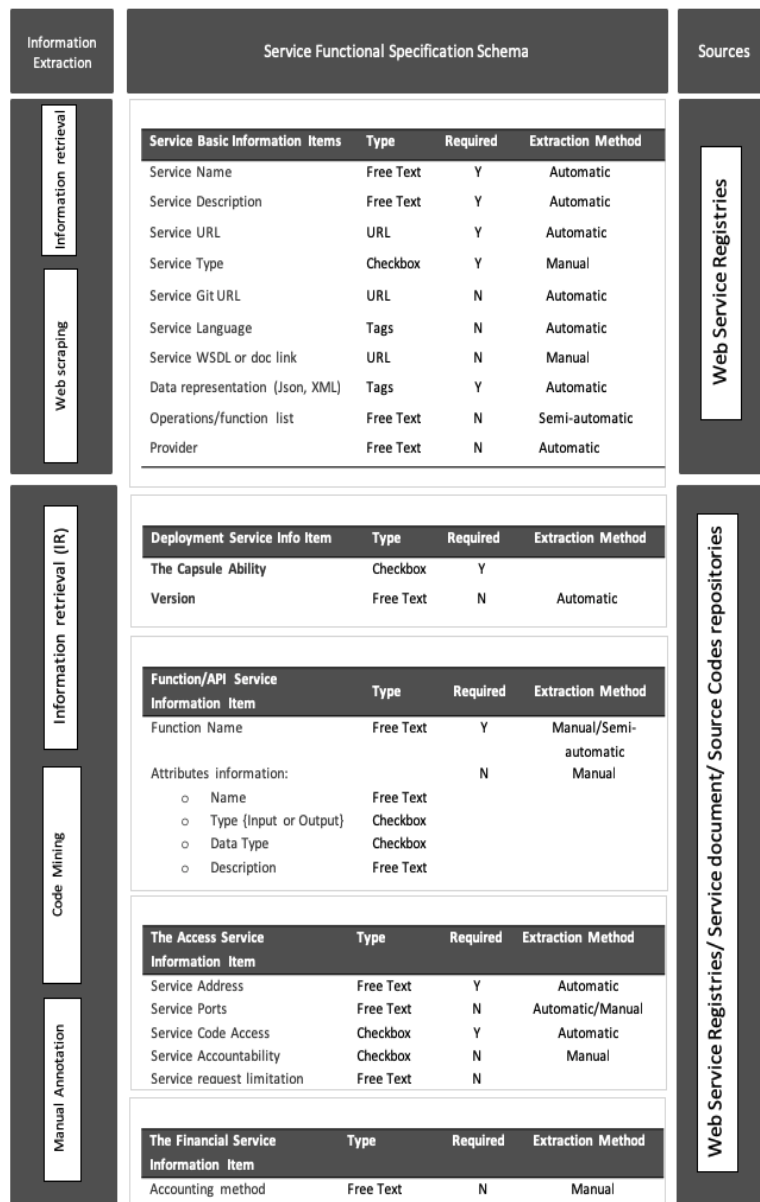


Figure 37: Service Functional Specification Schema

6.6 Service Specification Validation

In order to produce high quality service specifications, some form of practical validation should be done before they are being stored in a service registry. Validation in this context means ensuring that the required items in a schema specify what they are supposed to specify. The following definitions of validation levels for Web service specifications can be developed by SmartCLIDE. The intention here is to provide an extra tag for each service for providing a more advanced search. There are two levels that define the validation of the web service specification. The first level validates Service basic items and items data type. This validation will be provided by the meta-schemas (e.g., data type, required definition, minLength, maxLength) which can be defined by the self-descriptive JSON result. However, the second level provides a more strict validation which requires semi-automatic annotation, as well.

7 Code Generation

To improve productivity and software quality, a series of software development paradigms emerged to avoid programmer errors. One of the most popular paradigms was Model Driven Development (MDD) which, through a process guided by models, and in collaboration with other tools, allows the generation of code from these models. MDD reached maturity at the end of the 20th century using UML as one of its most advanced standards, although there are other modelling languages that can be used to the same level of satisfaction. Although this paradigm has experienced expansion and advancement, the software development industry still is unable to respond with the necessary speed to organisations that need to include these methodologies with the aim of changing their business models rapidly. Model-driven development is also known as Model Driven Engineering (MDE), from which different proposals are derived. The Model Driven Architecture (MDA) proposed by the Object Management Group (OMG) is possibly the best known design approach. However, there are other proposals such as Adaptive Object Models and Metamodels, which attempt to facilitate the construction of dynamic and adaptable systems. The latter strategy is highly related to business rules research, specifically when means are needed to describe business rules and automatically generate implementations.

The MDD paradigm covers a wide spectrum of research areas that should be taken into account such as modelling languages for model description, the definition of transformation languages between models, the construction of tools to support the tasks involved in modelling, the application of the concepts in development methods and in specific domains, among others. While the MDD proposes the use of a set of standards such as MOF, UML and QVT, which are very well documented and applied with great success, others have a long way to go in terms of definition and design. Due to the needs in the software industry to enable the integration of various software technologies in a fast and transparent way, as well as to adapt quickly to the changing needs of the business logic, the Model Driven Development (MDD) model has been adopted. In this development paradigm, a semi-automatic generation of software is carried out by using models. Under this approach, a business expert can express his or her knowledge in a formal modelling language and the IT team defines how it will be implemented. Besides, if desired, the same model can be implemented towards different platforms (Java, .Net, CORBA). This allows to generate several platform-dependent artefacts with less development effort, through the same platform-independent model, which represents the business knowledge [119].

Even though this paradigm presents numerous benefits, it also presents certain aspects that must be taken into account in order to allow its integration into the internal systems of any architecture that contemplates a long-term vision. In this work, a review of code generation under the MDD paradigm is carried out to produce complete multi-agent systems from code templates. For this purpose, a five-phase code generation model has been proposed, which uses a BPMN notation file for the specification of agents.

7.1 Code Generation Background

Monolithic applications have been used in traditional software architecture for decades. The problem is that managing reliability and scalability is difficult when they become larger. To this end, some software architecture approaches, like services-oriented architecture, have emerged. Regarding modern SOA complexity, several approaches based on model-driven, automatic code generation, and agent-based

models have been introduced. Model-driven approaches rely on high-level abstraction which is more understandable for humans. One of the promising Model-driven techniques is template-based code generation which uses model-to-text transformations. According to the literature, template-based code generation can fall into two categories: (1) Full-automatic software generation [18][148]; and (2) Semi-automatic code generation [23] [125].

First category's approach is unable to solve the particular problem, due to the fact that the process is succeeded only by using pre-defined templates. This kind of techniques have been used in web applications. In most web application frameworks, a code generator automatically generates the skeleton of the website and produces the code of some common portal site features. Although the process of code generation seemed to take root in web frameworks, now one can ascertain its existence into areas that really needed it. For instance, Benato et al. have suggested template-based code generation in self-adaptive software that can autonomously react to modifications in its execution setting [18]. Their generator module of source code, has 3 major levels: 1) Metamodel 2) Template Engine 3) Source Code. The template contains all the generator's logic module and uses new metamodel as its input.

In the second category, users need to define their expectations. BPMN editors are well-known visual programming tools. Moreover, low-code and no-code development can use template-based code generation utilizing visual modeling software. [155]. The idea behind these techniques is that complex code can be demonstrated as graphical blocks. These blocks can include UML diagrams [145], modules, services [73], etc. These black box items can represent the system's behavior in a simple and transparent manner. Moreover, a distributed application has different modules on several resources; therefore, automatic code generation can bring Complexity & Error Reduction and Consistency Enhancement. Regarding these features, service composition research has utilized them in order to generate code automatically [167]. MDA transformation entails transforming models from one level to another within the same system. The transformation between the two levels are automatic, which are divided into two main categories: 1) Model to Model 2) Model to Text. Automatic code generators are a type of model-to-text transformations.

Modern software development platforms are smart environments with more interactive behavior. To this end, multi-agent was mixed with MDA. A significant study by Ferber et al. combined multi-agent with MDA techniques. They introduced a model called AALAADIN which has the proposed model enable humans to meta-modelling concepts of groups, agents, and roles [51]. With emerging Agent-Oriented Software Engineering, several studies have investigated such as Challenges, and trends [166], Evaluation of agent-oriented methodologies [72], increasing integration by power-type-based language [57], etc. However, there is no agreement on which standard has to be used in this area of research. However, most of the works [17], [51], were based on UML and its extensions. By using multi-agent concept, researchers have been able to provide more efficient solution in distributed systems. Ivanova et al. [71] have introduced a solution based on MDA and composition models for Advanced Metering Infrastructure (AMI). AMI are complex systems that can provide communication between the control center and distributed agents. Therefore, techniques such as high abstraction modelling, reusability, adaptability are essential. To this end, their proposed model considers all of AMI system requirements such as self-describing, rich knowledge representation, service discovery [71]. Keogh et al. have proposed MAS Organisations considering runtime execution. The proposed model tries to provide acceptable flexibility by recognizing behavioural characteristics [79]. Ferber et al. have applied agent-based models in MDA in order to accelerate software development in multi-agent systems [51]. Also, Gomez-Sanz et al. have

introduced related tools for the INGENIAS [59]. The introduced tool provides a graphical editor which presents abstraction levels of items. However, it also provides code editors which allow users to customize their own code, as well. Moreover, it furnishes some recommendations during implementation for guiding users.

According to the literature, most proposed frameworks or tools for multi-agent systems are either very generic or presented for a specific purpose. As part of SmartCLIDE we propose a tool for code generation of complete multi-agent systems. By using SOA architecture, it brings multi-purpose ability and helps to support a wide range of third-party technologies and programming languages.

7.2 Template based code generation approach

In order to produce complete multi-agent systems from code templates, a five-phase code generation model is proposed, which starts from an agent specification file. This file will contain a definition of agents, and will be structured in an agent definition language such as BPMN; However, any structured text format (such as XML or JSON) that allows to define the agents individually and their connections, is also appropriate for this purpose. The essence of this language is to structure the elements used in this notation in a canvas using graphic tools. These elements can be divided into the following groups:

- Flow objects are the main elements, within of which we have events, activities and gateways.
- A template-based approach regarding the code generation within an agent paradigm
- Events describe events and are represented by circles. Depending on the type the outer circle can be simple (start), double (intermediate) or thick (end).
- Activities are the aggregation of tasks required for a program fraction that is subsequently added to the main program. They are represented by rectangles with rounded corners. If they are specific, they are indicated by an icon in the upper left corner.
- Gateways are control structures represented by diamonds.
- Connection objects relate each of the flow objects and can be sequences, messages or associations.

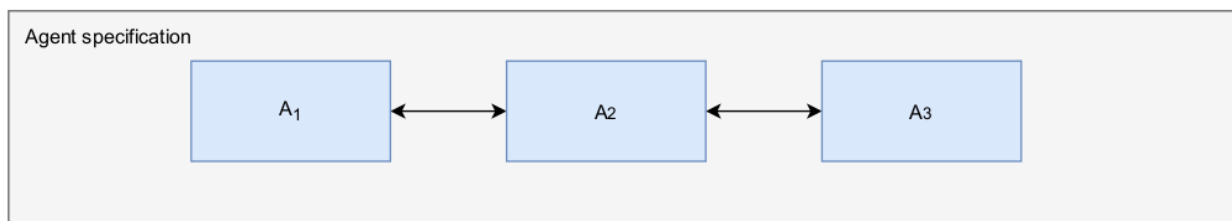


Figure 38: Agent specification in file

The agent specification file, is parsed in order to extract the structure with the information of the different events, objects and connections, to later generate a structured system from them. Therefore, to generate the multi-agent system's code, it is necessary to have a repository of metamodels, which will consist of code templates, whose parameters are provided in the agent specification file. The starting point for code generation is the agent specification file, and an example of the graphical representation of this file's content can be found in the 1. For the generation of this code, several phases have been defined and are described as follows:

- **Phase I:** The file is processed to obtain the list of objects modeled in it, such as agents, events, etc., as well as the information associated with them, such as parameter mappings between components, type

of service, associated template and other metadata of each component and event. This results in an object representation within a software application, formed by a flow of information between the agent specification, as shown in 2.

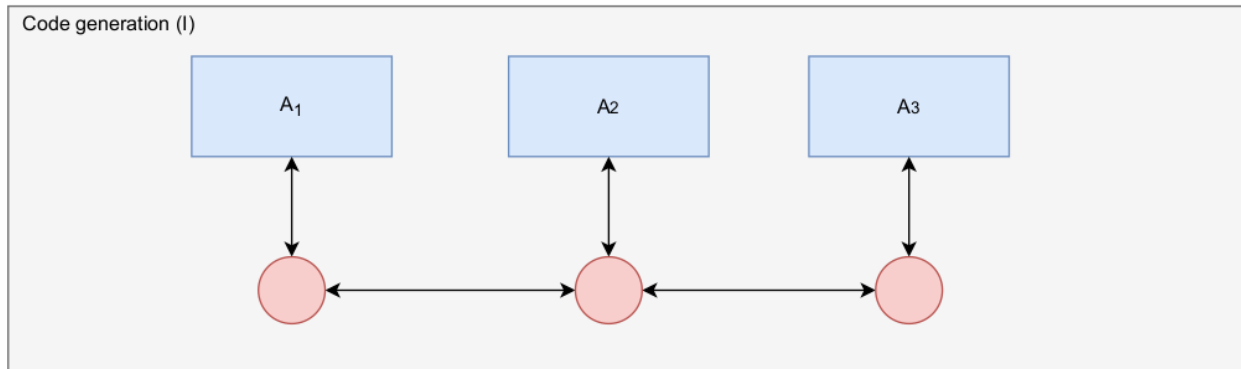


Figure 39: Phase I of code generation process

- **Phase II:** For each agent to be generated, the templates associated to each one are retrieved, then, the individual code of each one is generated. This results in an information flow as shown in 3, in which the functionality of each agent is already generated, but they are not communicated.

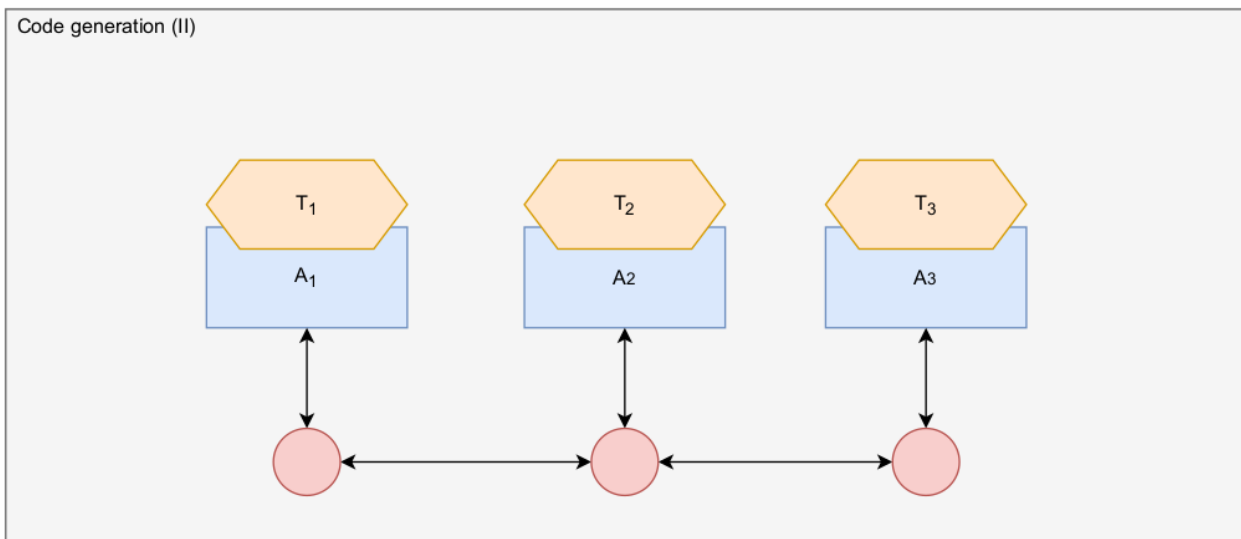


Figure 40: Phase II of code generation process

- **Phase III:** Each component is taken and also by means of templates, a protocol communication adapter is generated as interface within the internal behavior of the agent and the rest of the organization. It is worth to highlight, that in this case, due to the fact that the database of metamodels used for the tags is mostly made up of REST services whose functionality is to be incorporated into a multi-agent organization, HTTPS middleware has been used as protocol adapter. For the generation of this protocol adaptor, a template is also used, after which the parameter mapping is generated, resulting in a structure similar to the one shown in 4.

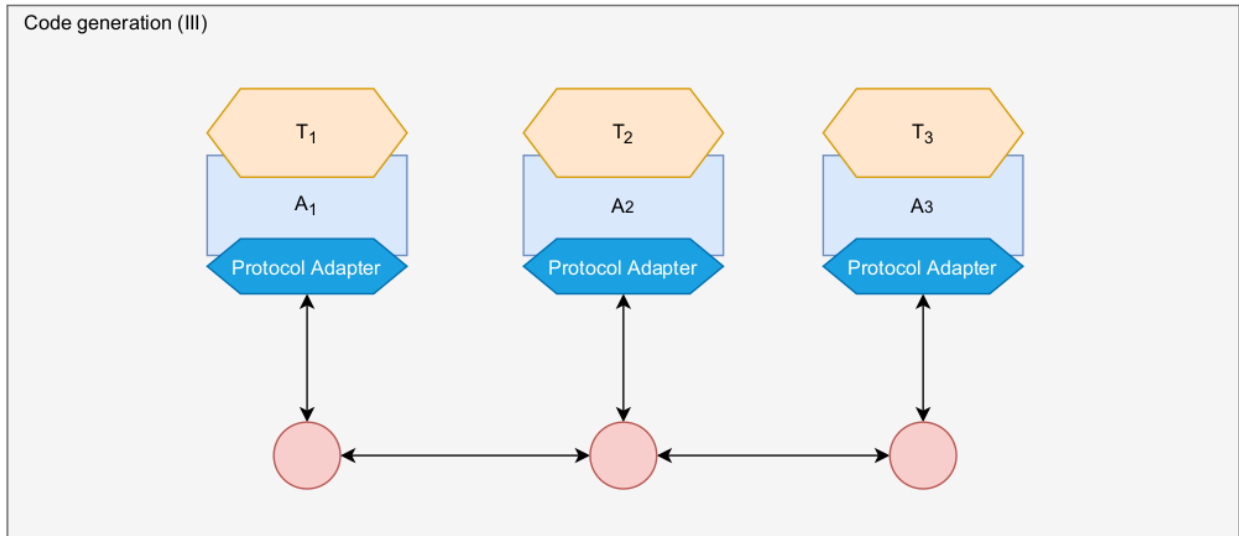


Figure 41: Phase III of code generation process

- Phase IV:** The communication schema defined in the agent specification file is used to generate the HTTP requests to each template from the interface offered by API rest generated in the previous step. Parameter mapping is also performed between each agent so that the output of one can be matched to the input of the next, resulting in a set of communicating agent as shown in 5. It should be noted that the programming language in which each template is written may vary, due to the protocol adapter.

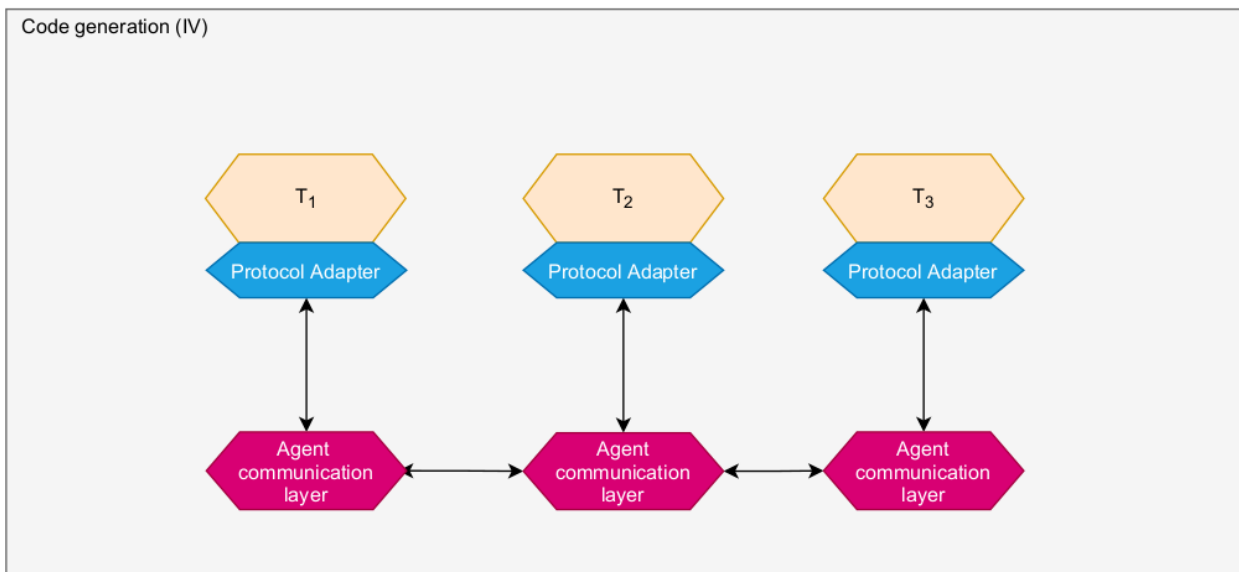


Figure 42: Phase IV of code generation process

The advantages of this code generation model is that it allows the integration of components from different programming languages into a multi-agent system, in order to communicate with a standard and efficient protocol, as long as the parameter mapping between each of the components (defined in templates) is specified.

7.3 Evaluation of generated systems

This section describes a code generation use case with the proposed system, focusing especially on the part of code generation and integration in the multiagent organization. For this purpose, a use case is proposed in which it is desired to create a multi-agent organization, composed of three agents. Each of them will perform a sentiment analysis of the same text with different techniques. Subsequently, by means of negotiation techniques, the result of the three agents will be composed in order to select the most appropriate one for the organization. For this end, three templates implementing sentiment analysis have been used, one for each agent, which have been described in Figure 43. The reference to the template and how to perform the mapping of input and output parameters has been established. In this file, other aspects have also been defined, such as the selected negotiation for the agents, which consists of agreement technologies [122]. However, other simpler ones have been implemented in the system, such as the composition by mean or median.

```
<?xml version="1.0" encoding="UTF-8" ?>
<organization>
  <meta>
    <name>01</name>
  </meta>
  <negotiator>argumentation-based-negotiator</negotiator>
  <agents>
    <name>A1</name>
    <template>a6dfed16-761d-47f0-adbe-6652508ad6d0-sentiment-analysis-1</template>
    <meta-template>
      <input-params>
        <name>text</name>
        <type>string</type>
        <required>true</required>
      </input-params>
      <output-params>
        <name>sentiment-compound</name>
        <type>dict</type>
      </output-params>
    </meta-template>
  </agents>
  <agents>
    <name>A2</name>
    <template>a6dfed16-761d-47f0-adbe-6652508ad6d0-sentiment-analysis-2</template>
    <meta-template>
      <input-params>
        <name>input_text</name>
        <type>string</type>
        <required>true</required>
      </input-params>
      <output-params>
        <name>sentiment-result</name>
        <type>array</type>
      </output-params>
    </meta-template>
  </agents>
  <agents>
    <name>A3</name>
    <template>a6dfed16-761d-47f0-adbe-6652508ad6d0-sentiment-analysis-3</template>
    <meta-template>
      <input-params>
        <name>t</name>
        <type>string</type>
        <required>true</required>
      </input-params>
      <output-params>
        <name>sentiment</name>
        <type>float</type>
      </output-params>
    </meta-template>
  </agents>
</organization>
```

Figure 43: Agent organization specification file

After applying the code generation with this input file, the code files corresponding to the three agents have been generated. In this article in the snippets (see Figures 44 and 45) are attached respectively an example of the result of the integration using the python API that wraps the sentiment analysis component of agent 1 and the java class that implements the integration with the analysis functionality of agent 1, since the agents are developed in JADE and communicate through ACL.

No examples of the code generated for the negotiation are attached, since it consists of another template derived from [32], which implements the selected negotiation.

```
import flask
from flask import request, jsonify
from a6dfed16_761d_47f0_adbe_6652508ad6d0_sentiment_analysis_1 import analyze

app = flask.Flask(__name__)

@app.route('/', methods=['POST'])
def behaviour():
    result = {}
    result['sentiment-compound'] = analyze(request.json.text)
    return jsonify(result)

app.run()
```

Figure 44: Generated HTTP rest adapter

```
public class AgentAdapter{

    public void adapt(String url, Map<String,String>params){
        OkHttpClient client = new OkHttpClient().newBuilder().build();
        MediaType mediaType = MediaType.parse("application/json");
        RequestBody body = RequestBody.create(mediaType, Utils.serialize(params));
        Request request = new Request.Builder().url(url).method("POST", body)
            .addHeader("Content-Type", "application/json").build();
        Response response = client.newCall(request).execute();
        JSONObject json = new JSONObject(response.body().string());
        String result = json.getString("sentiment-compound");
        return result;
    }
}
```

Figure 45: Java generated code from agent side

As a result of the deployment, a multi-agent organization is obtained, which is able to negotiate by obtaining the sentiment analysis of the provided text, reaching the result described in the Figure 46. The weights assigned in the figure described above have been negotiated with agreement technologies [60] by calculating the distance of the result to the average of all the results.

Agent	Sentiment result	Weight
A1	0.91	0.21
A2	0.83	0.67
A3	0.64	0.12
Total		0.824

Figure 46: Negotiation Results

7.4 Conclusions

The proposed system is capable of generating code for multi-agent systems focused on the integration of specific technology into one. This system generates the code using existing templates, customizing them by using a protocol adapter and creates the corresponding calls to integrate the generated components in JADE templates that communicate via ACL with a negotiation specified previously to the code generation. The result of this process is a multi-agent organization generated from templates that allows the use of a programming paradigm in multiple programming languages, facilitating the integration of these in the system. Among the future lines of work that will be exploited from this work, are the study through more use cases, incorporating a system of template ingestion to increase the existing database of templates and incorporating more algorithms for negotiation between agents.

8 Design Pattern Selection

8.1 Design Patterns Selection and Application

In this section we describe the research approach, introduced as part of the SmartAssistant, for supporting the application of design patterns. In Section 8.1.1 we focus on the most-known patterns catalogue, i.e., the GoF design patterns [56]. The GoF design patterns are established solutions for recurring object-oriented design problems that in the common case, are known for improving the extendibility and reusability of the source code [7]. Among the vast literature on GoF design patterns, one (rather neglected) line of research specializes on approaches that can aid in the application of design patterns. In Section 8.1.2 we present the “smart” approach for recommending the most suitable GoF design pattern (if any) for a given requirements / design problem, as well as the strategy for building a skeleton code that can drive the implementation.

8.1.1 State-of-the-Art Analysis

According to Ampatzoglou et al. [7], “*GoF design pattern application*” is the research sub-topic that involves research endeavours that present methods for identifying systems that need pattern application or methods and tools that automate or assist the application of patterns. The research landscape in this direction can be organized into 5 main categories: (a) design pattern abstraction; (b) re-engineering to patterns; (c) generative design patterns; (d) automated code transformation; and (e) pattern-based architecture. Each one of the aforementioned lines of research are described in detail below.

The *Design Patterns Abstraction* research topic includes studies (e.g., [21][80][160]) suggesting that design patterns are the key to provide abstraction in software and for adapting software components into existing systems. Bishop [21] presents how the use of the more abstract features of a programming language can decrease the gap between design patterns and their implementation. More specifically, Bishop [21] used as examples three design patterns (i.e., Bridge, Prototype and Iterator). Design patterns presents some of their own abstraction challenges: (a) the traceability of a design pattern is hard to maintain when programming languages offer poor support for the underlying patterns, (b) design patterns are used and reused in the design of a software system, but with little or no language support, developers must implement the patterns again and again in a physical programming language, (c) some design patterns have several methods with trivial behaviour, and without good programming tools, it can be more complicate to write all this code and maintain it, and (d) using multiple patterns can lead to a large cluster of mutually dependent classes, which lead to maintainability problems when implemented in a traditional object-oriented programming language.

Keepence and Mannion [80] develop a method that uses design patterns to model variability. The method starts by analyzing existing user requirements from systems within the family and identifying discriminants, which is any feature (requirement) that differentiates one system from another. There are three types for the identification of discriminants: (a) single discriminant, which is a set of mutually exclusive features, only one of which can be used in a system, (b) multiple discriminant which is set of optional features that are not mutually exclusive; at least one must be used, and (c) option discriminant which is single optional features that might or might not be used. The authors tested their method on ESOC’s spacecraft MPSs. They built a family user-requirement specification by editing and merging the

requirement specifications from three separate MPSs: ISO (a spacecraft that observes stars), ERS-2 (a remote-sensing spacecraft that monitors the earth's environment, and Cluster (a multi-spacecraft mission to monitor the earth's magnetosphere). The family user-requirement specification had 350 requirements (each MPS requirement specification had about 150 user requirements). Based on the analysis of the MPS family, they produced 20 class diagrams, 15 object-interaction diagrams, and 100 classes. This model lets developers identify and select desired features and build new family systems.

Yau and Dong [160] present an approach to apply design patterns to component integration. This approach uses a formal design pattern representation and a design pattern instantiation technique of automatic generation of component wrapper from design pattern. Design patterns are organized in a design pattern repository, where patterns are represented precisely using their design pattern representation. The design pattern representation should be expressive without jeopardizing the abstract feature of design pattern solution. Components and their descriptions can be retrieved from a component repository. The component description includes component interfaces expressed in IDL and semantics of services provided by components. After the selection of the design pattern, the pattern has to be instantiated to a concrete solution. Design pattern instantiation is to generate part of the software design, based on the generic solution in design pattern and application-specific pattern instantiation information. Finally, while applying design patterns, the designers should ensure the consistency between the original design patterns and the instantiated design patterns. The approach is assessed using an illustration example. The example is to develop a chatting room, which is used for several people in one group to talk simultaneously.

The *Re-engineering Anti-Patterns* research topic includes studies (e.g., [26][103]) that propose methods for detecting software anti-patterns that necessitate re-engineering through design pattern application. Briand et al. [26] present a structured methodology for semi-automating the detection of areas within a UML design of a software system that are good candidates for the use of design patterns. This is achieved by the definition of detection rules formalized using the OCL and using a decision tree model. More specifically, each tree corresponds to a design pattern (e.g., Decorator) or a group of design patterns when those patterns have strongly related structures and intent (e.g., Factory Method and Abstract Factory). Decision nodes in a tree denote a question in the decision process towards the identification of places in the design where design patterns could be used. When a series of questions have been answered, the tree leads to a decision where a design pattern is suggested. This corresponds to a path in the tree, from the root node to a leaf node. Additionally, some of the decisions are semi-automatic and involve user queries. Moreover, the authors illustrate their methodology using the Factory Method and the Abstract Factory Design Pattern. The aforementioned methodology has been implemented in a tool namely DPATool (Design Pattern Analysis Tool). The DPATool consists of three sub-systems: (a) the DPA Eclipse plugin, (b) the DPA Processing Engine, and (c) the DPA Model. The DPATool is a plugin to the Eclipse platform that interacts with two other Eclipse plugins, namely the Eclipse UML2 and Eclipse EMF plugins. The tool could be used by two different types of users. First, expert designers, who can define their own decision trees, for instance according to their observations of how designers in their organizations develop system. Second, every designer can be invoked whenever necessary during UML-based development support by Eclipse. To assess the feasibility of their methodology, they performed a case study of a test driver for an ATM. The ATM test driver has 15 classes with 114 operations and 45 attributes. The UML 2.0 models of the ATM test driver were reverse-engineered from the source code into the Eclipse platform. After processing the UML 2.0 model of the ATM test driver, the DPATool suggests the usage

of a Factory Method pattern. Also, DPATool suggested the use of the Visitor and the Adapter design patterns.

Meyer [103] provide an approach, which supports the detection of anti-pattern implementations in source-code. More specifically, the approach consists of three main steps: (a) anti-pattern recognition, (b) transformation, and (c) transformation verification. For the first step, the approach is based on an extended Abstract Syntax Graph (ASG) representation of a system's source-code. Anti-patterns are specified by graph grammar rules, which define as an ASG node structure which has to exist in the ASG representation and adds an annotation node to indicate the anti-pattern. The approach parses the source-code into the ASG representation and the anti-pattern rules are applied to the ASG by an inference algorithm. For the transformation step, the transformation rules are specified as graph grammar rules based on Story Diagrams. The software engineer manually examines the candidates identified by the first step and decides which transformations are to be applied to which candidate, if any. Then, the transformations are executed automatically in the transformed source-code. As the final step, the transformation rules must verify that the rules do not create forbidden or preserved anti-patterns.

Generative Design Patterns [95][96] correspond to techniques that aim to automatically generate design pattern instances. MacDonald et al. [95] present an approach to generative design patterns, trying to solve three problems: (a) there are no adequate mechanism to understand the variations in the source-code that spans the family of solutions and adapt the code for a particular application, (b) it is difficult to construct and edit generative design patterns, and (c) the lack of a tool independent standard. Their approach is independent of programming language and support tools. To validate the approach, they have implemented two tools, CO2P2S (Correct Object-Oriented Pattern-based Programming System) and MetaCO2P2S to support the process. The process consists of three steps. First, the software engineer selects an appropriate generative design pattern from a set of supported patterns. Second, he / she adapts this pattern for their application by providing parameter values. Finally, the adapted generative pattern is used to create object-oriented framework code for the chosen pattern structure. In a follow-up study, the same group [96] presents a design-pattern-based programming system based on generative design patterns that can support the deferral of design decisions where possible, and automate changes where necessary. Moreover, a generative design pattern is a parameterized pattern form that is capable of generating code for different versions of the underlying design pattern. Also, the author categorized the design decisions into two categories: (a) interface-neutral decisions—affect only the implementation of the structure of the pattern behind a stable interface, and (b) interface-affecting decisions—affect both the structure of the pattern and the framework interface to the application code. CO2P3S (Correct Object-Oriented Pattern-based Parallel Programming System, pronounced “cops”) generates Java frameworks for several common parallel structures, both shared-memory code using threads and distributed-memory code. The author demonstrated the capability of the system in the context of a parallel application written with the CO2P3S pattern-based parallel programming system.

The **transformation to pattern** research topic [68][110][111][112] includes studies that propose methodologies for automatically constructing transformations that can be used to apply GoF design patterns. O' Cinneide and Nixon [111] present a methodology and tool support, namely DPT (Design Pattern Tool), for the development of design pattern transformations. The methodology deals with the issues of reuse of existing transformations, preservation of program behaviour and the application of the transformations to existing program code. First, a design pattern is chosen that will serve as a target for the design pattern transformation under development. Then, the transformation is decomposed into a

sequence of mini-patterns (i.e., a design motif that occurs frequently across the design pattern catalogues). For each mini-pattern, a corresponding mini-transformation (i.e., an algorithm that applies the corresponding mini-pattern to the given program entities) is developed. Then, each mini-transformation should be demonstrated as a behaviour-preserving. The algorithm that describes the mini-transformation is expressed as a composition of refactorings. The final design pattern transformation can be defined as a composition of mini-transformations. The authors used the Factory Method transformation as an illustrative example. Moreover, the authors present a prototype software tool DPT that has been designed and implemented that can apply these pattern transformations to a Java program. Finally, they used an example of the application the Factory Method transformation to a generic program. The authors applied the methodology to a set of patterns from the Gamma et al. [56] catalogue, and prototyped the transformations. For each pattern, first the method finds a suitable precursor, assessing if a workable transformation can be built, and determining the mini-transformations that are likely to be used. Then, the authors assessed the results based on the three categories (excellent, partial, and impractical). The results suggest that half of the patterns have excellent transformation and 26% of the cases as partial.

Hsueh et al. [68] provide an approach for design pattern application and support the design enhancement by model transformation. For the selection of the pattern for the model transformation, the authors divided the pattern into six parts: pattern description, functional requirement intent, non-functional requirement intent, functional requirement structure, non-functional requirement structure, and transformation specification. For the automating pattern application, Hsueh et al. [68] document the refinement processes of patterns in regular rules and describe them in formal transformation language. Then, after specifying the transformation specification, they implement the mapping rules in ATLAS Transformation Language (which is a hybrid of declarative and imperative transformation language based on OMG OCL). For the evaluation of their approach, the authors performed a case study on a real-world embedded system PVE (Parallel Video Encoder). They define the Command Pipeline pattern to revise a sequential processing design to a parallel processing design in a generative TBB code.

Finally, Tonella and Antoniol [147] propose an approach for documenting design decisions in real-time, and enables *pattern-based architecture* through the inference of object-oriented (OO) design patterns from the source-code or the design. As a first step, the authors have used concept analysis to identify groups of classes sharing common relations. Next, the selected concepts containing maximal collections of classes having the same relations among them. The aforementioned concepts seem to be good candidates to represent design patterns inferred from the source-code or from the design. The number of instances of a pattern represents an indicator of the frequency of reuse of the identified class organization, while the number of involved relations represents the complexity of the pattern. To evaluate their approach, Tonella and Antoniol [147] performed a case study on C++ applications. They first examined the methods that owned by the involved classes. The results of their study suggest that the structural relations among classes led to the extraction of a set of structural design patterns, which could be improved with non-structural information about class members and method invocations.

8.1.2 GoF Design Patterns Application support in the SmartAssistant

This section will be completed in the final version of this deliverable, i.e., deliverable D.2.2

8.2 Architectural Patterns Selection and Application

In the context of the SmartCLIDE project it will be investigated how architectural patterns can be supported in a cloud IDE based on code templates/generators.

8.2.1 Architectural Patterns

The *layered architecture pattern* is one of the most common architectural patterns that matches the traditional IT organizational structures. As the name suggests, this architecture consists of components that are structured as horizontal layers, where each layer has a specific role within the application. Most layered architectures typically consist of the layers presentation, business, persistence and database. The presentation layer handles user interface and browser communication. The business layer handles the execution of specific business rules. The persistence layer contains the code to access the database layer, and finally the database layer is the underlying database technology.

Advantages of the layered architecture pattern include the following:

- High testability, due to the fact that components belong to discrete layers in the architecture, other layers can be mocked or stubbed, making this pattern is relatively easy to test.
- High ease of development because of the pattern's popularity and lack of complexity. Most companies develop applications by separating skill sets by layers. Thus, this pattern becomes a natural choice for most business-application development.
- It is maintainable.
- It is easy to assign separate "roles".
- It is easy to update and enhance layers separately.

The *event-driven architecture pattern* is a widely used architectural pattern that is adaptable for small and large applications. It consists of two main topologies: the mediator and the broker. The mediator is used to orchestrate multiple steps within an event through a central mediator. Within the mediator topology there are four main types of architecture components: event queues, and event mediator, event channels, and event processors. The broker is used to chain events together without a central mediator. Within the broker topology there are two main types of architecture components: a broker component and an event processor.

Advantages of the event-driven architecture pattern include the following:

- It is easily adaptable to complex, often chaotic environments.
- It scales easily.
- It is extendable when new event types appear.

The *microkernel architecture pattern* is a plug-in architectural pattern for implementing product-based applications. It allows you to add additional application features as plug-ins to the core application, providing extensibility as well as feature separation and isolation. It has two types of architecture components: a core system and a plug-in modules. The core system of the microkernel architecture pattern traditionally contains only the minimal functionality required to make the system operational. The core system is often defined as the general business logic sans custom code for special cases, special rules, or complex conditional processing. The plug-in modules are stand-alone, independent components that contain specialized processing, additional features, and custom code that is meant to enhance or

extend the core system to produce additional business capabilities. Generally, plug-in modules are independent of other plug-in modules.

Advantages of the microkernel architecture pattern include the following:

- It provides great flexibility and extensibility.
- Some implementations allow for adding plug-ins while the application is running.
- It allows for good portability.
- It provides ease of deployment.
- It allows for quick response to a constantly changing environment.
- Plug-in modules can be tested in isolation and can be easily mocked by the core system to demonstrate or prototype a particular feature with little or no change to the core system.
- It allows for high performance as one can customize and streamline applications to only include those features needed.

The *microservices architecture pattern* is an architectural pattern that is based on the concept of separately *deployed units*. Each component of the microservices architecture is deployed as a separate unit, allowing for easier deployment through an effective and streamlined delivery pipeline, increased scalability, and a high degree of application and component decoupling within the application. The most important concept of this pattern is the notion of a *service component*. These are components that can vary in granularity from a single module to a large portion of the application. Service components contain one or more modules (e.g., Java classes) that represent either a single-purpose function (e.g., providing the weather for a specific city or town) or an independent portion of a large business application (e.g., stock trade placement or determining auto-insurance rates). Designing the right level of service component granularity is one of the biggest challenges within a microservices architecture.

Advantages of microservices architecture pattern include the following:

- It allows to write, maintain, and deploy each microservice separately.
- It is easy to scale as one can scale only the microservices that need to be scaled.
- It is easier to rewrite pieces of the application because they are smaller and less coupled to other parts.
- New team members must quickly become productive.
- The application must be easy to understand and modify.
- Highly maintainable and testable – enables rapid and frequent development and deployment.
- Independently deployable – enables a team to deploy their service without having to coordinate with other teams.

Cloud (space-based) architecture pattern is specifically designed to address and solve scalability and concurrency issues. It minimizes the factors that limit application scaling. This pattern gets its name from the concept of *tuple space*, the idea of distributed shared memory. There are two primary components within this architecture pattern: a *processing unit* and *virtualized middleware*. The processing-unit component contains the application components (or portions of the application components). This includes web-based components as well as backend business logic. The virtualized-middleware component handles housekeeping and communications. It contains components that control various aspects of data synchronization and request handling.

Advantages of the cloud/space-based architecture pattern include the following:

- It responds quickly to a constantly changing environment.

- Although cloud architectures are generally not decoupled and distributed, they are dynamic, and sophisticated cloud-based tools allow for applications to easily be “pushed” out to servers which simplifies deployment.
- High performance is achieved through the in-memory data access and caching mechanisms build into this pattern.
- High scalability come from the fact that there is little or no dependency on a centralized database, therefore essentially removing this limiting bottleneck from the scalability equation.

Service-oriented architecture(SOA)

- service granularity: can range from small, specialized services to enterprise-wide services
- a collection of services that are able to communicate with each other
- communication: each service must share a common communication mechanism called an enterprise service bus (ESB)
- each service is the endpoint of a connection
- each service is comprised of the code and data integrations required to execute a specific business function
- loose coupling, i.e. services can be called with little or no knowledge of how the integration is implemented underneath
- interoperability: heterogeneous messaging protocols such as SOAP (Simple Object Access Protocol), AMQP (Advanced Messaging Queuing Protocol) and MSMQ (Microsoft Messaging Queuing)
- reusability of integrations (reusability and component sharing)
- synchronous calls: the reusable services in SOA are available across the enterprise using predominantly synchronous protocols like RESTful APIs
- governance: shared resources, enable the implementation of common data governance standards across all services
- storage: a single data storage layer shared by all services within a given application

8.2.2 Non-Functional Requirements

Non-functional requirements that play role in determining the architecture pattern for an application including the following ones:

Scalability refers to the capacity to maintain effective performance during a steep increase in workload without the need to redesign the software architecture. This can be, for example, a significant increase in the number of users using the application. A general approach to scaling an application can be *horizontal scalability* (scale out), which adds or removes nodes to a system, and *vertical scalability* (step up), which adds or removes resources to a node in a system. Three commonly used systematic approaches to scale an application is as follows: *Scaling the x-axis* by horizontally cloning the application, *scaling y-axis* by splitting functionality and *scaling the z-axis* by partitioning or sharding the data.

Reliability refers to the consistency in the anticipation of software operations. A common metric to measure reliability is the number of software faults (bugs), expressed as faults per thousand lines of code. The best way to mitigate bugs in software is to follow a good testing strategy, where a software architecture that is easy to test can potentially become more reliable than other architectures that are harder to test.

Performance refers to the **amount of work** accomplished by a system and means the limiting factor in the end usability of the system. Is totally dependent of the needs of the project, but good performance involves **different means**: For UI refers to a *low latency*, less execution time, at server code refers to *high throughput*, that is the rate of processing of work and in embedded architectures refers to low utilization of *computing resources*.

Availability is the capacity of a software architecture to be *fully or partially operational* when is required, and to effectively handle failures that could affect it. Some tactics and actions related to availability are high-availability hardware and load balancing, fault-tolerant software, component replication, zero downtime deployments, design for failure, backup and disaster recovery solutions. A service oriented architecture like microservices can use *load balancing* between instances and ensure more availability than a client server architecture with server as a *single point of failure*.

Usability refers to how easy it's for a user to use the platform and achieve the expected results with **effectiveness, efficiency** and **satisfaction**. Usability includes the ability of being accessible to all people, even those with **disabilities**. This can be accomplished implementing assistive technology in the UI, and providing text for all images and multimedia resources.

Security is a process of risk management that balances likely *security risks* against the **cost** of guarding against them. This requirement allows the owners of resources in the system to **reliably control** who can perform what actions on particular resources. It implies many actions, including sensitive **resources** identification and protection, identification and **authorization, availability** protection, secure **libraries**, security **administration**, information **integrity** and continuous **monitoring** of vulnerabilities and threats.

Reusability in software architecture refers to the degree in which parts of the architecture can be used in other software systems. It brings **cost reduction** due to the reuse of known and tested software components. Service oriented architecture is a good candidate for reusability, e.g. a **microservice** can be reusable in different systems.

Simplicity implies reduce responsibilities through **software decomposition** and separation of concerns using clear **interfaces**. The main goal of simplicity is to reduce complexity, this contributes to more maintainable and less costly systems. A **centralized architecture** will be simpler than a **distributed architecture** because the inherent communications that distributed environment requires.

Flexibility(adaptability) refers the ability for the software architecture to adapt to possible or future changes in its requirements. A software architecture is flexible when its parts have *low coupling*. When a system can evolve with only adding things, it says that system is **very flexible**. Conversely, if it is needed to change architecture parts in order to evolve it, then it's not flexible. For example, layered architecture brings a lot of flexibility because parts of architecture are **clear separated** and can be evolved.

Portability refers to the degree in which the same architecture can be used in different environments. One key factor element of portability is a clear abstraction between business logic and system interfaces.

Maintainability is about how easy the software system can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment. Some **metrics** used to measure maintainability include technical debt, code smells, cyclomatic complexity, source lines of code. Another key factor to ensure maintainability is *mean time to repair*, the average time required to repair a specific item or component from notification of a failure until to return it to working status.

Testability refers to the degree to which a software part of architecture supports testing in a given test context. High testability implies more facility to find software faults or bugs that improves reliability.

Supportability (serviceability) refers to the ability of a software system to ease **technical support** in some areas including **installation** and configuration of the system, logging or tracing, monitoring health metrics of the application.

Concurrency refers to the capacity of handle multiple computations executing simultaneously, and potentially interacting with each other. A software architecture with high concurrency have clearly identified parts that can execute concurrently.

Cost is one of the important non-functional requirements of a software project. There are a lot of elements to consider in the overall cost of a software architecture. The most outstanding examples include: **Implementation** cost, costs associated to **failure management** and recovery, **data backup** strategy costs, **resources** needed to run the system, costs associated to system **maintenance** and support.

Life time is related to **system development life cycle** in which a software system goes through a series of **life cycle** phases such as planning, requirement analysis, design, development, integration and testing, deployment and maintenance and disposal. In order to **discontinue** a software system, there is a process of transition to a new system. Design architectures that take this into account will facilitate **migrations** in the future.

8.2.3 Workflow

- The user starts creation of a new project/application.
- The *User Frontend* component asks the user to provide input, specifically non-functional requirements, further information about the project.
- The *User Frontend* presents the user a list of supported architectural patterns to choose from optionally.
- The *Architectural Pattern Inference* component receives the input provided by the user via the *User Frontend* and infers possibly a set of architectural patterns that are most suitable for the project.
- The set of inferred architectural patterns are presented to the user.
- After the user selects the preferred architectural pattern(s) the *Architectural Pattern Application* component finishes the creation process by generating the project and configuration scaffold.

User Frontend The *User Frontend* component asks the user to provide input, specifically non-functional requirements, further information about the project including type of applications (enterprise, event-based, product-based, web-based).

The *User Frontend* presents the user a list of supported architectural patterns to choose from optionally.

Architectural Pattern Inference The purpose of the *Architectural Pattern Inference* component (APIC) is to help the application developer choose one or a combination of possibly two architectural patterns that are most appropriate for her application. APIC proposes patterns by analysing the non-functional requirements and other information related to the application that are provided by the application developer during her interaction with the *User Frontend* component.

Table 10: NFRs supported by Architectural Patterns

Pattern	Non-Functional Requirements
Layered Architectural Pattern	Maintainability, simplicity, consistency, flexibility, portability, testability, cost
Event-driven Architectural Pattern	Agility, reliability, scalability, extensibility
Mikrokernel Architectural Pattern	Scalability, extensibility, portability
Microservices Architectural Pattern	Reusability agility, scalability, portability, testability, availability, resilience
Cloud-based Architectural Pattern	Availability, scalability, resilience
SOA	Reusability, testability, availability

Criteria that are used in the inference process for each of the architectural patterns:

Layered architecture is best suited for the following situations:

- New applications that need to be built quickly
- When application does not need to scale out and business logic is not very complex
- Teams with inexperienced developers
- Enterprise or business applications that need to mirror traditional IT departments and processes
- Applications requiring strict maintainability and testability standards.

Key features: Separation of concerns

Event-driven architecture is best suited for the following situations:

- Applications that have asynchronous data flow systems.
- For complex applications that require seamless data flow or those applications that would eventually grow.

Key features: Asynchronous data flow systems, loose coupling

Microkernel architecture is best to use for the following situations:

- Applications that require or are concerned with the separation between low-level functionalities and higher-level functionalities.
- Since the microkernel pattern provides extensibility, scalability, and portability, it is best used for enterprise applications.
- It is best suited for development teams that are spread out.

Key features: Plug-in components

Microservices architecture

- service granularity: highly specialized independent services
- communication: asynchronous communication
- loosely coupled
- interoperability
- does not enable consistent data governance
- code reuse
- service specific data storage

- cloud-native architectural approach, often operating in containers
- agile, scalable, portable, resilient

Cloud architecture is best for the following situations:

- For applications and software systems that work under a heavy load of users that access or write to the database concurrently
- For applications that need to address and solve scalability and concurrency issues.
- Best suited for e-commerce or social website development.

Architectural Pattern Application The Architectural Pattern Application component (APAC) completes the creation process by generating the project and configuration scaffold.

8.3 Security Patterns Selection and Application

8.3.1 Introduction

Given a problem statement and a set of forces acting on the system, design patterns instruct its stakeholders on how to build a system. Patterns in the information technology environment provide information system architects with a technique for creating reusable solutions to design challenges without ever having to discuss or write program code; they are really programming language-independent.

The goal of employing patterns is to produce a design element that can be reused. Each design has a purpose in and of itself. The combination of patterns enables people responsible for implementing security in producing good, consistent designs that incorporate all needed processes, ensuring that the ensuing implementations can be performed efficiently and effectively [22].

Information system security is becoming more crucial than ever. Companies are increasingly reliant on information systems, therefore software along with the information that it manages have become immensely commercially vital. Data integrity can be jeopardized, for example, if data is destroyed or changed by malicious individuals. Competitors are given access to business information. Furthermore, compromised systems may drastically harm a company's image and reputation, often leading to devastating financial losses.

One of the issues we encounter in everyday practice is the need to protect an application without expending excessive time and effort; as a result, we are inclined to adopt well-known methods such as configuring a firewall or employing basic password authentication. Applying a pattern, a solution that has previously been substantially utilized in practice, may appear to be a logical approach. However, in many circumstances, a solution implemented without a full knowledge of security needs does not provide effective protection within the unique environment [82].

The goal of the work presented in this section is to provide an overview of the main Security Design Patterns that have been proposed in the literature over the years and that are widely used in practise. Specific emphasis on the dependencies that exist between the different security patterns is also provided. In fact, Patterns are typically used in groups of Security Design Patterns. The dependencies and relationships that various patterns have with one another will be highlighted. Another important goal of the present section is to showcase how the envisaged SmartCLIDE platform will attempt to assist developers in selecting the most suitable security patterns and implementing them in their software.

In brief, the performed analysis that is described in the present section, is a fundamental part of the solution that will be implemented as part of the SmartCLIDE platform, with respect to security patterns selection.

In the rest of the section, Security Design Patterns will be discussed briefly at first, with information on additional reading and dependencies on other patterns provided. The broad dependency diagram will then be sketched out to show how Security Design Patterns are interconnected. Then an overview of the envisaged application will be also provided.

8.3.2 Requirements for Security

The purpose of the Security Design Patterns is to help software developers and engineers improve the overall security of the produced software. The overall security of a given application can be decomposed into a set of broader security qualities/requirements that need to be satisfied. The most widely used qualities and security features in the security domain will be used in this part; the established criteria are based on the works of Babar [12] and Firesmith [53].

The following security attributes have been chosen:

- *Authentication*: The system must validate the identity of its externals before interacting with them. Customers' identities must be authenticated in order to prevent unauthorized access.
- *Authorization*: Authorized externals' access and usage privileges are correctly given and enforced. This property establishes an entity's access privileges to various system resources and services.
- *Integrity*: There should be a method in place to safeguard data from unauthorized change when it is held in an organizational repository or moved over a network. It should guarantee that active assaults do not damage data and communications.
- *Confidentiality*: Sensitive information is not given to other parties who are not authorized to receive it (e.g., individuals, programs, processes, devices, or other systems). A system should protect data and communication privacy against unwanted access. The concealment of resources is a key part of secrecy.
- *Attacker detection*: Attempted or successful assaults (or the harm caused by them) are detected, recorded, and reported. It entails being able to identify and register unauthorized users' access or modification intents in the system.
- *Non-repudiation* is achieved when a party to an interaction (e.g., communication, transaction, data transmission) is precluded from effectively repudiating (i.e., denying) any component of the interaction. It precludes a participant in an engagement from denying that they took part in it.
- *Security auditing*: By studying security-related events, security staff can audit the state and usage of security systems. This entails preserving a record of how users or other systems interact with a system. It aids in the detection of possible attacks, the investigation of what occurred following assaults, and the gathering of evidence of unusual activity.
- *Maintainability*: It permits the implementation or adjustment of security policies across the software development life cycle.
- *Availability* ensures that authorized users have access to resources when they are needed. It guarantees that authorized users may access data and other resources without hindrance or disruption. It assures that a system recovers swiftly and thoroughly in the event of a calamity.

8.3.3 Security Patterns based on Security Requirements

Security Patterns are abstract representations of actual solutions that could be employed for satisfying specific security requirements, and therefore for strengthening the overall security of an application. Design Patterns (in general) are classified into three types according to their level of abstraction [27] (high-level, mid-level, and low-level):

1. An architectural pattern is a basic structural organization schema for software systems that expresses a basic structural organization schema. It defines a collection of predefined subsystems, describes their duties, and offers rules and principles for structuring their interactions.
2. A design pattern is a method for improving the subsystems or components of a software system, as well as the interactions between them. It depicts a recurrent pattern of interacting components that solves a broad design challenge within a specific environment.
3. An idiom is a low-level pattern that is unique to a certain computer language. An idiom specifies how to use the characteristics of a given language to implement certain attributes of components or connections between them.

A pattern, according to Ramachandran [126], is a typical and recurrent style of solution design and architecture. A pattern, according to Ramachandran, is a solution to a problem in the context of an application. To the expense of other purposes, security components tend to focus on hardening the system against danger. Patterns add balance to the description of security architecture since they emphasize both good architecture and excellent security.

Our choices of security features, authentication systems, and access control models can either push our architecture toward a well-understood design pattern or lead us to an ad hoc solution with significant architectural tensions. If we choose the latter approach without a model for security architecture, we may find weaknesses or hazards in the solution's creation only after deployment.

Security patterns provide ways for recognizing and resolving security vulnerabilities; they collaborate to establish a set of best practices (to support a security strategy), and they handle host, network, and application security. Patterns have several advantages: they may be examined and adopted at any moment to enhance a system's design; and less experienced practitioners may benefit from the expertise of those who are more fluent in security patterns. They give a consistent language for discussion, testing, and development, as well as the ability to conveniently search, categorize, and refactor them. They provide security techniques that are reusable, repeatable, and documented. They don't specify programming techniques, languages, or suppliers [19].

For certain application security circumstances and restrictions, design strategies decide which application tactics or design patterns should be employed. Safety is paramount. Design patterns are abstract representations of business challenges that satisfy a wide range of security needs and give a solution to a known security issue (s). They might be architectural patterns that show how to handle a security challenge structurally, or they may be defensive design principles that secure code may be built atop afterward [144].

An **architectural pattern** is a list of element and relation types, as well as a set of restrictions on how they can be utilized. A pattern may be conceived of as a collection of restrictions on an architecture - on the element types and their patterns of interaction - that create a set or family of architectures that fulfill those constraints. Client-server architecture, for example, is a typical architectural pattern.

Client and server are two element kinds, and the protocol that the server employs to interact with each of its clients is used to characterize their coordination. This (loose) concept encompasses a large number of architectures, all of which are distinct from one another. Although an architectural pattern is not a building, it nonetheless offers a good image of the system by imposing appropriate limitations on the architecture and, as a result, on the system [16].

For software systems, an architectural pattern describes a basic structural organizing schema. It outlines the duties of established subsystems and offers rules and guidelines for arranging the interconnections between them [144]. A high-level abstraction is an architectural pattern [27]. A crucial design decision in the construction of a software system is the architectural pattern to be employed. It establishes the overall structure of the system and limits the design options accessible to the constituent subsystems. It is, in general, unaffected by the implementation language being utilized. Broker, multi-layer, pipe and filter, transaction processing, and other architectural patterns are examples.

One of the most helpful features of patterns is that they display well-known quality characteristics. This is why the architect choose a certain pattern rather than one at random. Some patterns are well-known solutions to performance issues, while others are well-suited to high-security systems and have been successfully used in high-availability systems. Choosing an architectural pattern is frequently the first important design decision made by an architect [16]. Some popular architectural patterns that are used for enhancing security are described below:

- **Data Filter** pattern [54] filters in a distributed system the contents of client requests based on specified policies. Filtering might take place on-site or remotely. Requests for services or data in many distributed systems, such as the Internet, must be filtered according to institution policies, statutory limits, privacy considerations, and so on.
- **Check Point** design [154] [163] centralizes and enforces security policies while also encapsulating the mechanism for putting the policy into action. Any number of security checks can be included in the algorithm. This pattern may also be used to keep track of unsuccessful security breach attempts, which can aid in taking appropriate action if the failures are malicious. An example of the Check Point pattern is provided in Figure 47.

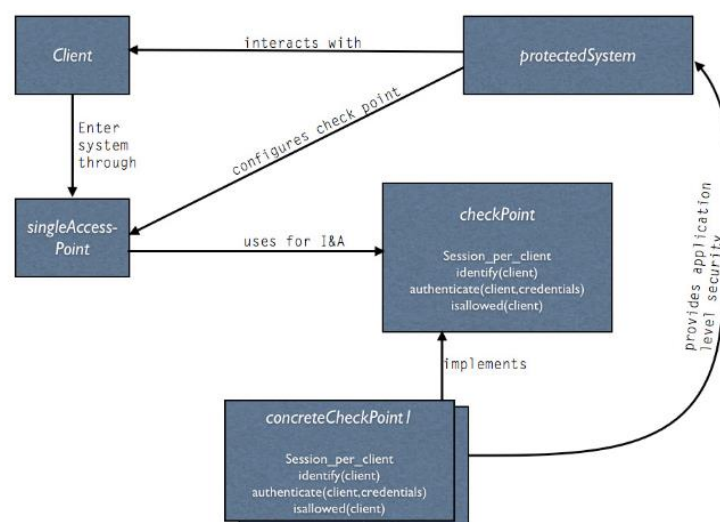


Figure 47: Check Point Flow Diagram [133]

- **BodyGuard** pattern [44] enables us to share and regulate access to objects in a distributed context without the need for system-level distributed object support. The Bodyguard pattern is a network management design that makes it easier to control object sharing. To prevent inappropriate access to an object in collaborative applications, it enables message dispatching validation and assignment of access permissions to objects in non-local contexts.
- **Layered Security** pattern [131] aims to divide a system's structure into numerous levels in order to increase the system's security by safeguarding all of the layers. One of the biggest disadvantages of employing this pattern is that it adds to the architecture's complexity.
- **Encryption** [137]: There is nothing quite like a system that is completely secure. An attacker may be able to obtain access to a system and its data. As a result, the sensitive information included in this data should be safeguarded. The term "obscurity" must be used with caution. The phrase "information obscurity" is used in the book by Schumacher and Sommerlad [137] to denote "obscuring information by encrypting it." On the one hand, this goes against the idea of "no security via obscurity," but on the other hand, hiding the location of information is often done solely to make it more difficult to obtain it.

Design Patterns. Like patterns in any other subject, mature software design patterns capture solutions that have developed and grown through time. As a result, they are not the first designs that come to mind. Developers have strived for more reuse and flexibility in their program, and mature patterns reflect many cycles of untold redesign and recoding. Design patterns include sophisticated solutions in a concise and easy-to-apply format.

The goal of employing patterns is to produce a design element that can be reused. Each design has a purpose in and of itself. The combination of patterns aids those in charge of security implementation in producing good, consistent designs that contain all needed processes, ensuring that the ensuing implementations can be executed quickly and successfully [22].

A design pattern is a method for improving a software system's subsystems or components, as well as the interactions between them. It refers to a recurrent structure of communicating components that addresses a broad design challenge in a specific setting [27]. A design pattern is a type of intermediate abstraction. The choice of a design pattern has no impact on the software system's fundamental structure, but it does have an impact on the structure of a subsystem. The design pattern, like the architectural pattern, is usually independent of the implementation language to be utilized. Some popular security patterns that are used for enhancing security are described below:

- The *Audit Interceptor* pattern [144] works in combination with the Secure Logger pattern to offer instrumentation of the logging components in the front. In addition, this architecture facilitates administration and maintains logging and auditing in the backend.
- The *Secure Logger* pattern [144] explains how to collect application-specific events and exceptions in a secure and reliable way so that security audits may be performed. For better understanding, an example of the Secure Logger pattern is provided in Figure 48.

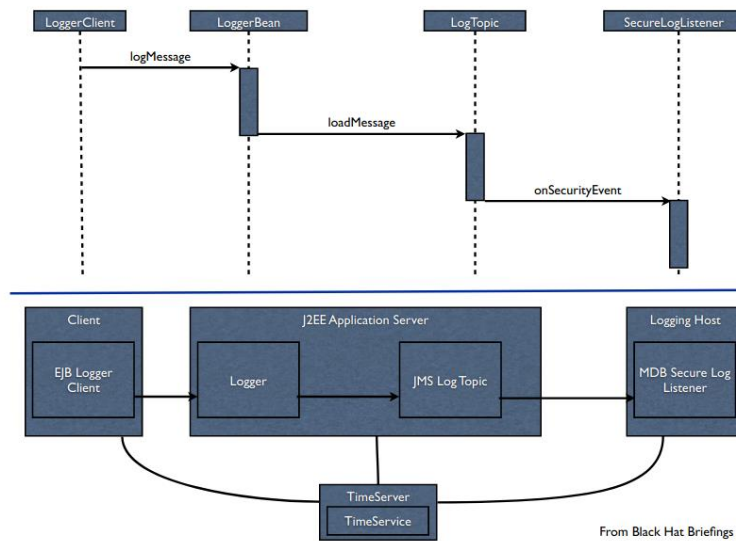


Figure 48: Secure Logger Flow Diagram [133]

- The *Assertion Builder* pattern [144] describes how to construct an entity assertion (for example, an authentication or permission assertion).
- The *Secure Pipe*: When linking trading partners, this pattern [144] shows how to secure the connection between client and server, or between servers. There will be a mix of security needs and limits between clients, servers, and any intermediates in a complicated distributed application environment. By demanding reciprocal authentication and establishing secrecy or non-repudiation between trading parties, it increases value. This is especially important when utilizing Web services for B2B connectivity.
- *Authoritative Data Source* pattern [130] is used to authenticate the legitimacy and origin of data. It protects the system from employing out-of-date or erroneous data, as well as reducing the possibility of false data being processed and spread.
- *Authenticator* pattern [88] is a generic way for a client to provide identify and authentication to a server. It also has the benefit of allowing protocol negotiation to be done using the same techniques. The pattern works by presenting an authentication negotiation object, which only provides the protected object after successful authentication. For better understanding, an illustration of the Authenticator pattern is provided in Figure 49.

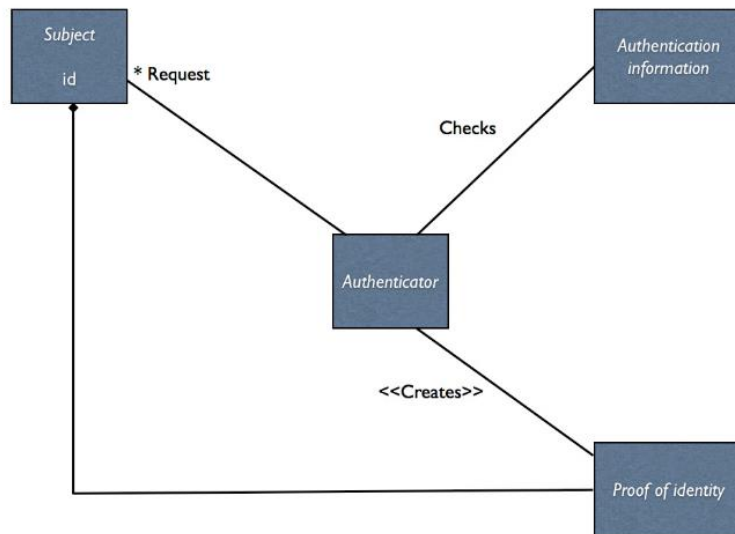


Figure 49: Authenticator Flow Diagram[133]

- *Session*[154][163]: Rather than re-authenticating a user for each system function, this pattern provides a way to retain track of a user's global information and deliver it to the systems functions without having to transfer all access privileges or re-authenticate a user. It generates a session object associated with the user or process requesting access to the system. For better understanding, an illustration of the Session pattern is provided in Figure 50.

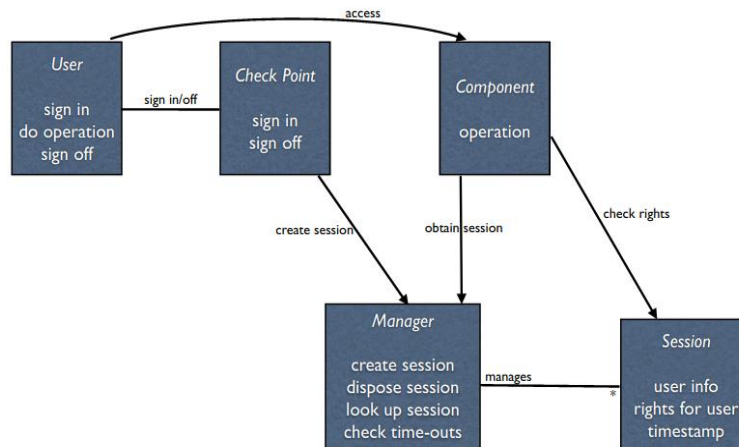


Figure 50: Session Pattern Flow Diagram [133]

Relation Between Patterns and Requirements. As already stated, security patterns (both design and architecture) may exhibit interdependencies. These interdependencies are highly useful, as far as the selection of a subset of suitable security patterns is concerned. Knowing the existing interdependencies can help a developer and engineer better decide which patterns to include in their implementation. Hence, a research on the dependencies between the security patterns was considered necessary. The results of our analysis are presented in Table 11. More specifically, Table 11 shows the relationship between security needs, security architectural patterns, and security design patterns that may be used to ensure that our

design based on these patterns meets and assures the security needs for the system built. We chose a subset of the aforementioned security requirement categories; we investigated many security patterns that drive and lead us toward secure development as well as a security software architecture based on security patterns. In brief, Table 11 maps security patterns (both architecture and design) to those Security Requirements that are related to, and that could be used as solutions for implementing these requirements into the produced software.

Table 11: Requirements, Architectural and Design Patterns [132]

Security Requirements	Architectural Patterns	Design Patterns
Authentication	Data Filter[54], Check Pont[154][163], SSO[81], Cryptographic[137]	Authenticator[88], Assertion Builder[144], SSO Delegator[144]
Authorization	Data Filter[54], Bodyguard[44], Check Pont[154][163], Firewall[44], Cryptographic[137]	Assertion Builder[144], Session[154][163]
Confidentiality	Firewall[44], Check Pont[154][163], Cryptographic[137], Encryption[137], Layered Security[131]	Secure Pipe[144], Session[154][163], Information Secrecy[25], Multilevel Security[44]
integrity	Firewall[44], Layered Security[131], Cryptographic[137], Encryption[137], Data Filter[54]	Message Integrity[25], Session[154][163], Multilevel Security[44]
Non-repudiation	Cryptographic[137], Encryption[137]	Secure Pipe[144], Signature[25]
Audit	Check Point[154][163], Single Access Point[154][163]	Audit Interceptor[144], Secure Logger[144]

To help the reader better understand the importance of the relationships between the different security patterns, some indicative examples are provided in the rest of this section. For instance, for the security requirement ‘Authorization’, the security administration that arranges the approval prerequisite is the Authorization Service (Transport level and Application level), and we know a few examples for fostering the referenced help as the structural examples Firewall, BodyGuard, Check Point, and so on that apply the engineering vital for the completing of the approval and the security configuration designs Authorization, Session, and so on, that carry out somehow or another, the assistance needed by satisfying the chose security necessities type. As a result, we will be able to construct and build the architecture and processes that attain and fulfil the desired need type if we employ these security patterns.

Another important security requirement that could be considered is the Confidentiality of the sensitive data that the application handles. Modern cryptography is extensively utilized in a broad range of applications, including word processors, spreadsheets, databases, and electronic commerce systems. The widespread usage of cryptographic techniques, as well as the current interest in and study on software structures and patterns, lead us to develop cryptographic software structures and cryptographic patterns. This architecture is made up of various patterns that provide cryptographic services to meet the needs of applications. These patterns are focused on the security qualities of confidentiality, integrity, authentication, and non-repudiation. The Information Secrecy pattern shows how to protect

communications from an attacker (confidentiality). The Message Integrity pattern demonstrates how to prevent an attacker from modifying or replacing messages without the exchange of cryptographic keys. The Sender Authentication pattern demonstrates how communications may be authenticated through the use of cryptographic keys. The Signature pattern illustrates how communication parties might be prevented from repudiating a communication (non-repudiation). Based on these generic cryptographic patterns, we can generate more patterns by combining one pattern with another, resulting in a single pattern that meets multiple security requirements. For example, the Secrecy with Integrity pattern is the result of combining the patterns Information Secrecy and Message Integrity, where the properties of confidentiality and integrity are guaranteed at the same time.

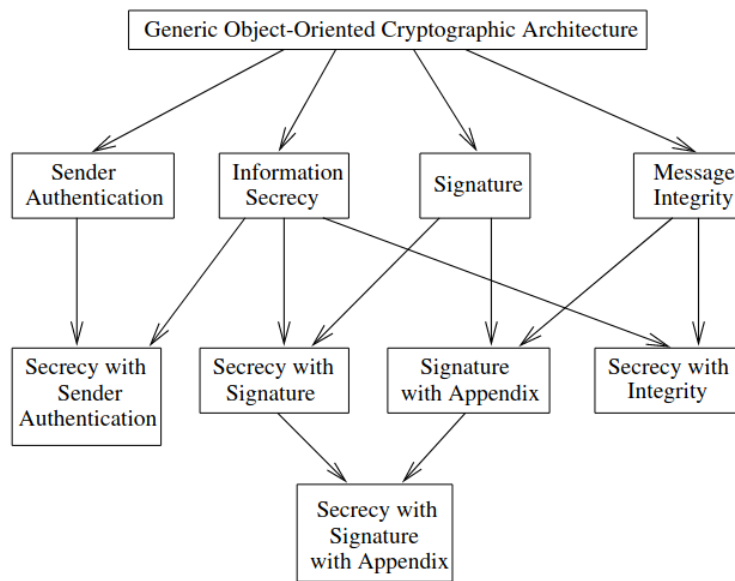


Figure 51: Relationships between cryptographic design patterns [25]

In [25], an architecture based on these patterns is defined as Generic Object-Oriented Cryptographic Architecture (GOOCA), which is an abstraction of all these patterns together to produce a generic architecture. Figure 51 depicts a directed acyclic network of pattern dependencies. An edge connecting pattern A and pattern B demonstrates that pattern A yields pattern B. A pattern that is pointed at by more than one edge has the same number of generators as the number of edges that arrive in it. The GOOCA creates the microarchitecture for the four fundamental patterns. All additional patterns are created by combining these two.

This is most likely one of the most common combos. The SINGLE ACCESS POINT is deployed in most systems in conjunction with the CHECK POINT, which some AUTHORIZATION and AUTHENTICATION patterns hook into. The patterns involved SINGLE ACCESS POINT, CHECK POINT AUTHENTICATOR, AUTHORIZATION, SECURITY SESSION, ROLE-BASED ACCESS CONTROL, METADATA-BASED ACCESS CONTROL, MULTILEVEL SECURITY and REFERENCE MONITOR.

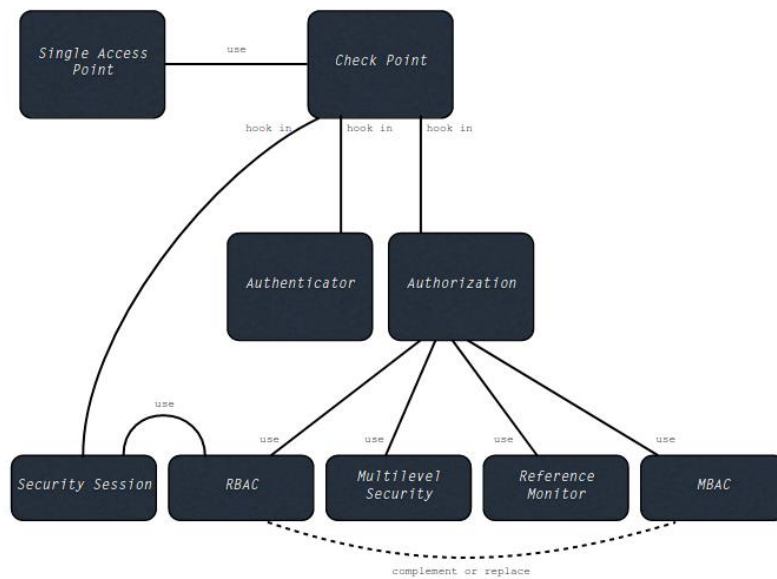


Figure 52: Block diagram of a typical composition/pattern around the CHECK POINT pattern

Figure 52 depicts a block diagram of the patterns employed. The SINGLE ACCESS POINT serves as the foundation, ensuring that all communications pass via this point. At this stage, the CHECK POINT is activated, which enables AUTHENTICATION and AUTHORIZATION functions. At the checkpoint, an SECURITY SESSION is generated upon successful authentication and authorization. The AUTHORIZATION pattern itself employs various more specific patterns, including the ROLE-BASED ACCESS CONTROL, METADATA-BASED ACCESS CONTROL, MULTILEVEL SECURITY, and REFERENCE MONITOR. The multilevel-security pattern is used to categorize system resources depending on their security level, whilst role-based access control specifies users' access rights to the various security levels of the resources. In circumstances when there is no direct role behind the process, metadata-based access control can be used to map process-rights based on their properties to resource-levels. Finally, the reference monitor is used to intercept all requests and validate access privileges, whether they are role-based or metadata-based.

8.3.4 Security Patterns into Software

Software companies, in order to enhance the security of their products, must meet specific security requirements, as mentioned above. Security patterns, whether design or architectural, provide solutions both during product design and during product implementation to meet the security requirements to the greatest extent possible within a system. They are abstract descriptions from solutions that can be implemented in the code to improve the overall security. Usually, when designing a product, companies develop security strategies at a later stage, resulting in the lack of implementation of security policies and requirements. By adopting architectural patterns from the beginning and combining the appropriate design patterns, greater fulfilment of the security requirements in software products can be achieved.

Over the years, as programming languages and frameworks have evolved, software mechanisms have been developed to implement architectural and security patterns. Each programming language and structure is accompanied by security libraries, with ready-made solutions or solutions that need

modification to adjust system criteria to meet the security prerequisites in a software product. We researched technologies and structures that are widely used to integrate these standards into software and link security requirements to language libraries. The results of this survey are presented in Table 12.

Table 12: Mapping between Security Requirements, Security Patterns, and actual frameworks/libraries that can be used for their implementation into software

Security Requirements	Security Pattern	Java/Spring Framework	Python/Django
Authentication / Authorization	Authenticator	JAAS, Apache Shiro, Spring Security Oauth, Google Auth	Python Oauth 2.0 library, Django OAUTH Toolkit, FastAPI, Google Auth
	Assertion Builder	AssertJ, Spring Framework	unittest
	SSO Delegator	OAuth2Authentication, Spring Security	Python-saml, django-simple-ssso
	Session	Spring Security Oauth, HttpSession	requests, Python Oauth 2.0 library
Confidentiality	Cryptographic	Spring Security	requests
	Firewall	IPTables	nfqueue
Integrity	Cryptographic	Spring Security,	requests, cryptography, ECDSA
	Firewall	IPTables	nfqueue
	Message Integrity	JSSE, ECDSA, URLClassLoader	hashlib, integrity-check
Non-repudiation	Cryptographic	Spring Security, javax.crypto	requests, cryptography, ECDSA, rsa
Audit	Secure Logger	Spring Data JPA, JPA, Hibernate	Auditing, PyAudit, django-easy-audit
	Audit Interceptor	Spring Data JPA, JPA, Hibernate	Auditing, PyAudit, django-easy-audit

The information presented in Table 12 is highly useful from a practical viewpoint. Based on the analysis presented in this section, the selection of the most suitable security patterns that should be utilized by the developers and engineers in order to enhance the security level of their applications is not a trivial task. First of all, it requires security expertise in order to know the existing patterns and which security requirements they strengthen, as well as the interdependencies among the existing patterns. It requires also important technical expertise in order to know which are the most popular frameworks/libraries that are commonly used for their implementation for specific programming languages, and a lot of effort for their actual implementation in the source code. Hence, a mechanism able to encapsulate this security

expertise and assist developers and engineers select the most suitable security patterns that they need to adopt based on the security requirements that they would like to satisfy, and also provide recommendations of which specific frameworks/libraries can be adopted for their implementation is necessary in practice.

To this end, such a mechanism will be implemented as part of SmartCLIDE. More specifically, regarding the SmartCLIDE solution it is important to have a mechanism that could accept as input the security requirements that the developer wants to satisfy and return a list of standards related to the requirements that he wants to fulfil, what are the most common technologies it can use based on its requirements and what are the most common libraries that could be used to implement the selected standards based on the system requirements (programming language). Additionally, the mechanism could also provide low level information (where feasible), e.g., code samples, code templates, etc., in order to further assist the developers in the correct implementation of the security patterns into the source code. In Figure 53 a high-level overview of the envisaged solution is illustrated.

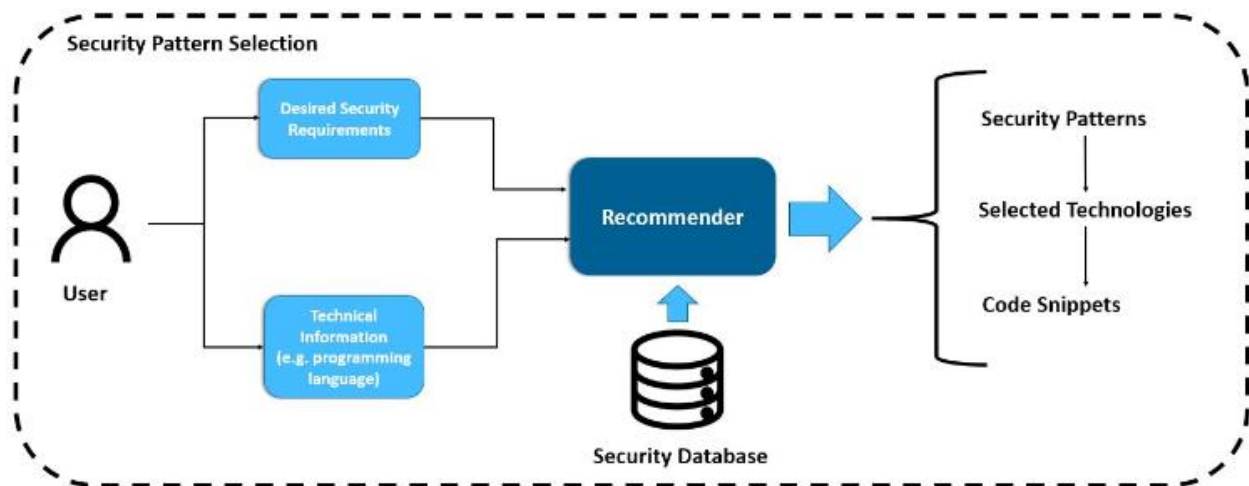


Figure 53: A high-level overview of the mechanism for selecting security patterns and suitable technologies

As can be seen in Figure 53, as an input the user should declare the security requirements to be satisfied by his product, such as Authentication, Authorization, Integrity etc. Moreover, since the mechanism will be language specific, the user input must also contain the implementation language of the product. As a next step, a search in a database based on user input is performed. This database will contain a language-specific list of technologies for each pattern and code snippets to provide a code-ready template for each pattern the user wants to implement. The proper technologies are then chosen from the database and presented to the user as a list of suggested technologies for each security requirement. Finally, after picking the proper technology, code snippets will be presented to the user in order to assist the developer to attach the selected patterns properly inside his implementation.

To better understand the aforementioned mechanism, an example is provided. Let's assume that the user wants to include Authentication and Authorization mechanisms as a Security pattern into his product. The user should declare the security requirements as well as the language of the implementation (Java for our example). After the search in the database, the list of the suggested technologies will be presented (Basic Auth, Token Auth etc.) along with information about them. A list of the language-specific libraries and frameworks that can be used for the implementation of the displayed technologies will be also presented

to the user. After picking the proper technology from the suggested ones, information about them (guidelines, tutorials, etc.) will be presented, and, if available, a code snippet will be presented as an output, in order to further assist the developer understand how the pattern could be realized into the actual source code of the software application. Let's assume that the user selects Basic Auth with Java API, then the code snippet could be something like the following:

```

@Configuration
@EnableWebSecurity
public class CustomWebSecurityConfigurerAdapter extends WebSecurityConfigurerAdapter {

    @Autowired
    private MyBasicAuthenticationEntryPoint authenticationEntryPoint;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("user1").password(passwordEncoder().encode("user1Pass"))
            .authorities("ROLE_USER");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/securityNone").permitAll()
            .anyRequest().authenticated()
            .and()
            .httpBasic()
            .authenticationEntryPoint(authenticationEntryPoint);

        http.addFilterAfter(new CustomFilter(),
            BasicAuthenticationFilter.class);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

```

Figure 54: Example of code snippet corresponding to Basic Auth with Java API

To sum up, the proposed mechanism is expected to help software developers and engineers with little to no security expertise, determine (i) which security patterns should be implemented in the software for satisfying desired security requirements, and (ii) which specific technologies should be utilized for their implementation, providing also further assistance for their actual implementation. This is expected to further enhance the security level of a software application, since critical security patterns will be selected and adopted from the early stages of the software development lifecycle, whereas through the recommendations of this mechanism the selected solutions are expected to be implemented correctly, avoiding the introduction of vulnerabilities.

9 Service Composition and AI

9.1 Workflow Functionality Summarization from BPMs

9.1.1 Proposed Approach

In this section we present a novel approach for the automatic summarization of BPMN-based workflows to a textual description including the type, purpose and basic characteristics of the involved process. The availability of textual descriptions that summarize workflows in the context of SmartCLIDE, beyond the documentation purpose that it serves, can be valuable in improving service and workflow discovery, as the author input (in the form of keywords or phrases) can be mapped to existing service compositions through textual analysis. The proposed approach relies heavily on the use of Natural Language Processing (NLP) techniques.

Text summarization is the automated process of parsing individual phrases or pieces of text and extracting a meaningful synopsis. Such processes are gaining ground since the amount of available information in the Internet era is enormous and continuously updated, rendering the human analysis extremely time consuming. In most cases, text summarization aims at automatic classification of texts, news and posts aggregation, automate title generation, etc. Nevertheless, it is a challenging task since the way humans comprehend and synthesize text is subject to numerous parameters which are hard to model in an algorithmic way.

One domain where text summarization proved to be promising is the automated generation of documentation for source code. Often developers write code under extreme time pressure hindering the introduction of internal documentation in the form of comments, thereby reducing the comprehensibility and maintainability of existing codebases. Employing text summarization approaches for yielding the summary of a method or class considering as input the tokens of the source code and method names can lead to quite meaningful documentation.

The proposed approach receives as input any existing process model represented in the form of an XML (BPMN) file, parses and analyses it and returns a summary in English (see Figure 55). The functionality is delivered in the form of a web service so as to be easily accessible by third systems but it also available to the end user who can easily upload the files to be processed. Since calls are independent of each other, simultaneous text summarization for multiple processes is supported.

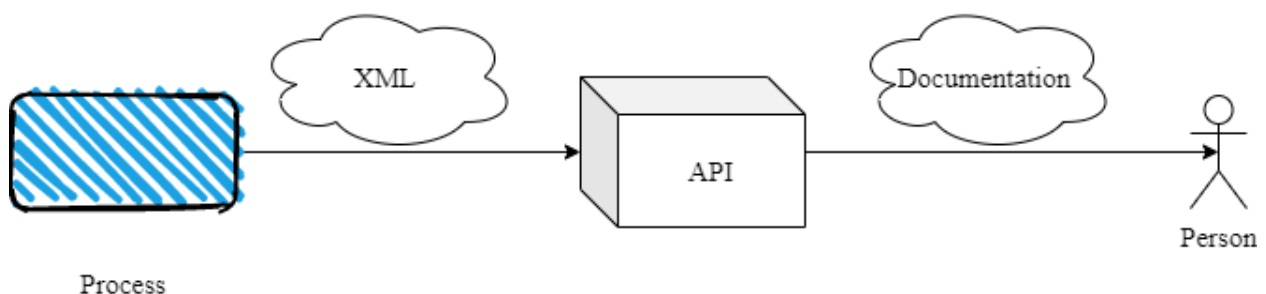


Figure 55: High-level process of text summarization

The server-side application has been developed using the Java Spring Framework and particularly the Java Spring Boot version. Communication with other systems is performed through REST calls and the HTTP protocol. The internal architecture is outlined in Figure 56.

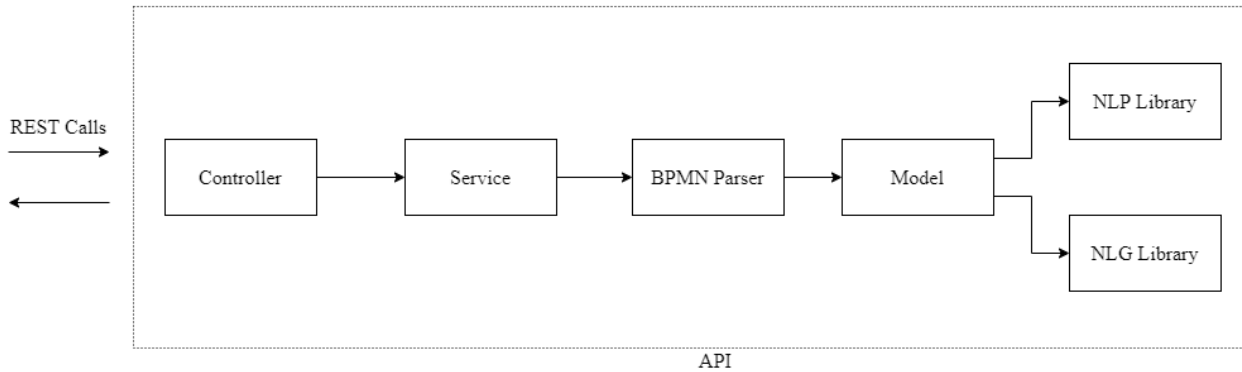


Figure 56: Internal Architecture for the proposed Text Summarization service

- The *Controller* accepts all calls from other systems and returns a response when the task is complete. Essentially it defines the available services and provides a first level of validation in the communication with other systems ensuring that the process has been terminated properly.
- After a successful call to an entry point (*Controller*) a *Service* is invoked which handles the request.
- The *BPMN parser* contains methods for breaking down the process to individual pieces (xml parser), understands the workflow and consolidates individual documentations to generate the final summary text.
- The *Model* contains classes representing all elements of the target business process model.
- *Natural Language Processing* (NLP/NLG) is responsible for the extraction of information (such as the name of an element, inputs/outputs, type of node etc.) from the BPMN elements to be used in the summary text.

During the parsing of the entire process specific characteristics are sought in the workflow. From these characteristics certain adjectives can be extracted with constitute the generic description. The next table summarized concepts which are being used.

Table 13: Generic Characteristics of the Workflow

Characteristic	Workflow
Scheduled	Scheduled process in case it receives a timer event.
Automatic	No human user is involved in the process
Single approval	One stage of approval
2-level approval	Two stages of approval
3-level	Three stage approval
Multiple-level approval	More than three user approvals
Short (single task)	One task
Short (2-task)	Two tasks

Characteristic	Workflow
Short (3-task)	Three tasks
Flat (without branches)	The flow does not contain any branches
Multi branched	Large number of branches
Alternate ending (N-ways)	Multiple alternative endings (end-events).
Repetitive	Presence of a circular flow

9.1.2 Indicative Results

Let us consider the simplified process of a book reservation shown in Figure 57.

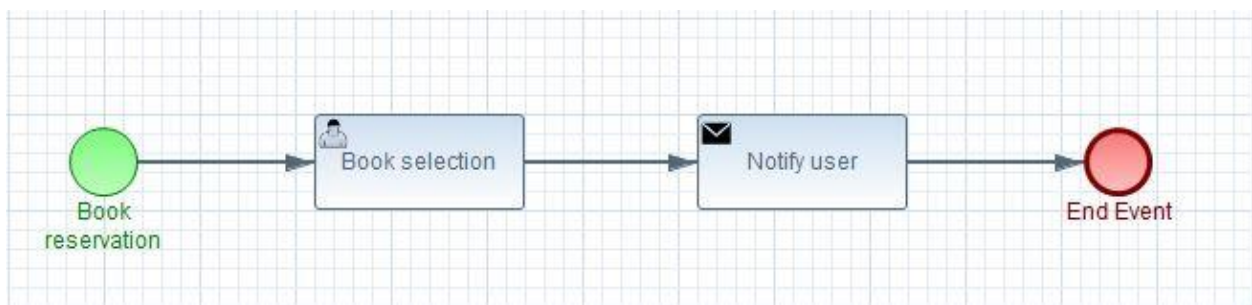


Figure 57: Simplified input process

The text summary which is automatically extracted reads as follows:

This is a single approval, short (2-task), flat process about book reservation. A user decides about the book selection. A send task is used to notify user. Then the process ends.

With green highlighting we denote the generic description from which it can be deduced that a single approval by the user is performed, that it is a flat process without alternative paths in the flow, with two individual tasks and refers to the reservation of a book. The process type is retrieved from the start event. With blue highlighting the two tasks are summarized based on their chronological order. Finally, yellow highlighting designates the ending of the process.

An example with a branching is illustrated in Figure 58. Right after the selection of a book it is checked whether a copy is available or not. If yes, the reservation is completed successfully; otherwise the process is terminated.

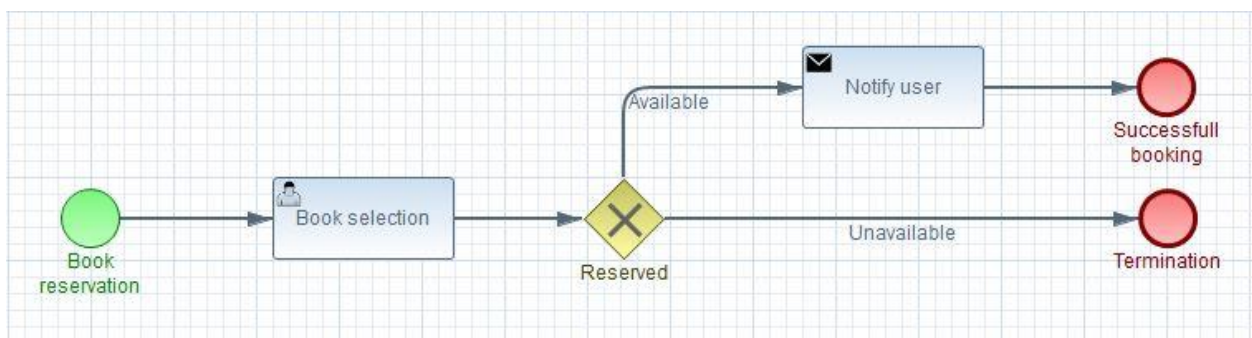


Figure 58: Input process with branching

The corresponding text summary that has been automatically extracted by the proposed application is:

This is a single approval, short (2-task) process about book reservation. A user decides about the book selection. A decision is taken depending if reserved.

If available: A send task is used to notify user. Then the process ends.

If unavailable: Then the process ends.

While the generic description is almost similar to the previous one, the characterization as ‘flat’ has been removed and there is a branch in the process.

A more complex example with two branches is shown in Figure 59.

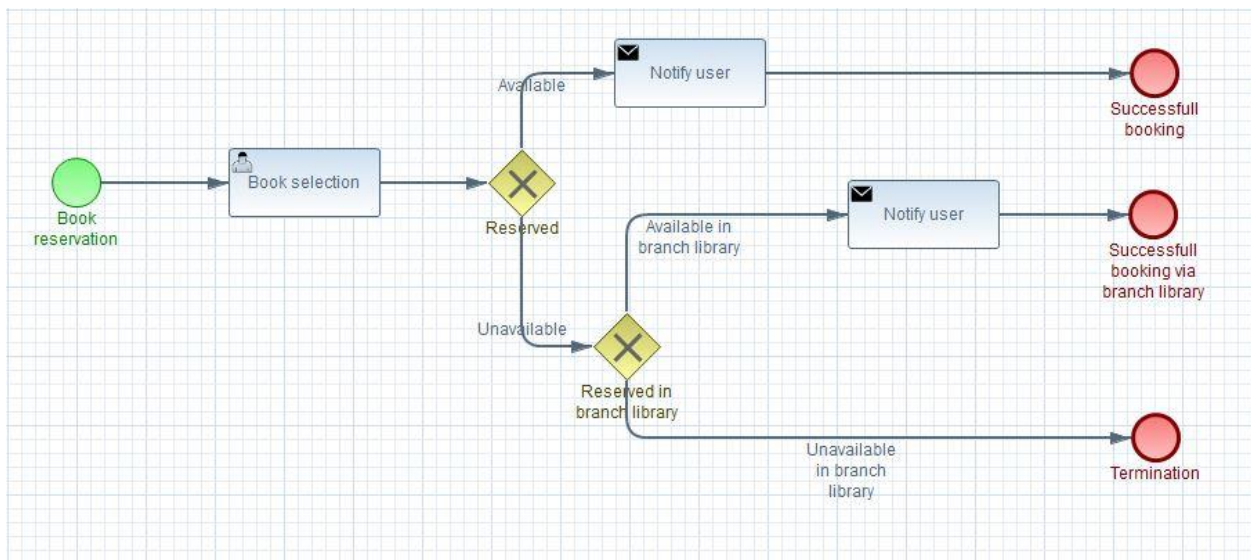


Figure 59: Input process with two Gateways

The corresponding text summary is:

This is a single approval, short (3-task), alternate ending (3-ways) process about book reservation. A user decides about the book selection. A decision is taken depending if reserved.

If available: A send task is used to notify user. Then the process ends.

If unavailable: A decision is taken depending if reserved in branch library.

If available in branch library: A send task is used to notify user. Then the process ends.

If unavailable in branch library: Then the process ends.

The process description varies depending on the elements present in the workflow. For example, Figure 60 depicts a process with the use of a timer.

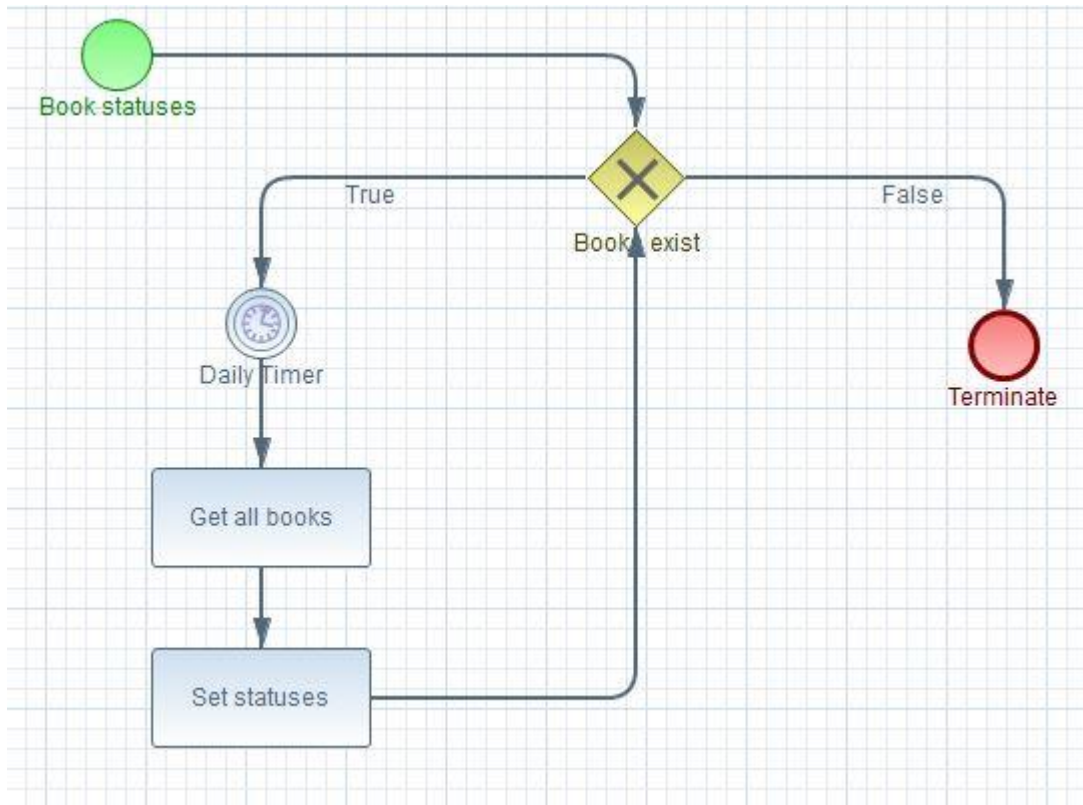


Figure 60: Input process with a Timer

The text summary is modified accordingly:

This is a scheduled, automatic (not involving any human task), short (2-task), repetitive process about book statuses.[156]A decision is taken depending if books exist.

If true: A daily timer is used to repeat the following process. A method is called to get books. A method is called to set statuses. The same flow is repeated [156].

If false: Then the process ends.

9.2 Service Composition Autocomplete Suggestions

This section presents a novel approach related to Smart Assistant functionality in service composition. This AI-based approach provides recommendations during service composition which are based on a selected approach based on data representation, existing/history BPMN work-flows, and provided service specification information. The suggestion system consists of automatically selecting the next step in the business process being modelled. For this, the recommender system relies on information either from a pre-existing database or gathered from the user's current composition process, i.e. the user's current context. Based on this, the model autonomously predicts the next step.

There are two main methodologies for defining suggestion systems:

- Collaborative filters: Based on the principle that similar users will have similar opinions about similar products (or whatever is being recommended).

- **Content-based filters:** These are based on identifying alternatives of potential interest based on similarity to elements already identified by the user as being of interest.

In the context of SmartCLDIE both techniques are going to be used for recommendation of new nodes in BPMN diagrams. Collaborative filters will analyse the BPMN diagram that is under development, and content-based filtering will find nodes with similar semantics to be provided as suggestions. The system will integrate a knowledge base for the different users, collecting their decisions during the construction of BPMN diagrams. When a user is working on the construction of a new BPMN diagram, the knowledge bases of the most similar users can be integrated, thus approaching the collaborative filtering scheme. On the other hand, when issuing the recommendation, the most interesting alternatives will be selected according to this integrated knowledge base.

Based on the analysis work done on a dataset of 2268 BPMN files extracted from Github, Bitbucket and Github we have designed a simple scenario that captures the main needs for recommending services during the design of BPMN workflows. The adopted solution will be executed as a back-end component in SmartCLIDE and will be receiving as input a recommendation request. This request can be a one-off signal from the SmartCLIDE monitoring system when a step in the workflow diagram drawing is completed as shown in Figure 61. The output signal, in JSON format, will contain the recommended service to link to the node indicated in the request. That is, it will not have recommendations at all times, only when the DLE so determines.

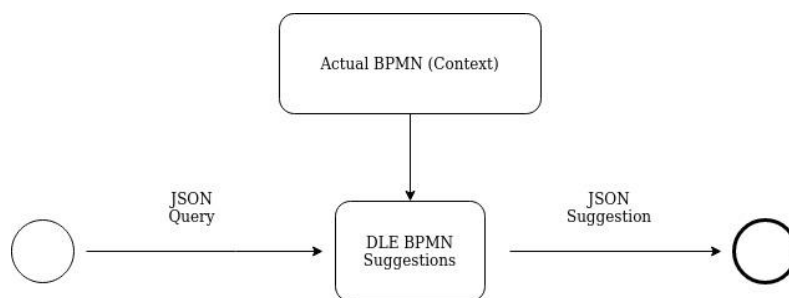


Figure 61: High level approach of Service Composition suggestions

The suggestion system for the BPMN design will be implemented in Python and will be queried via REST API in JSON format. The details of the internal stages of the DLE BPMN Suggestions are depicted in Figure 62.

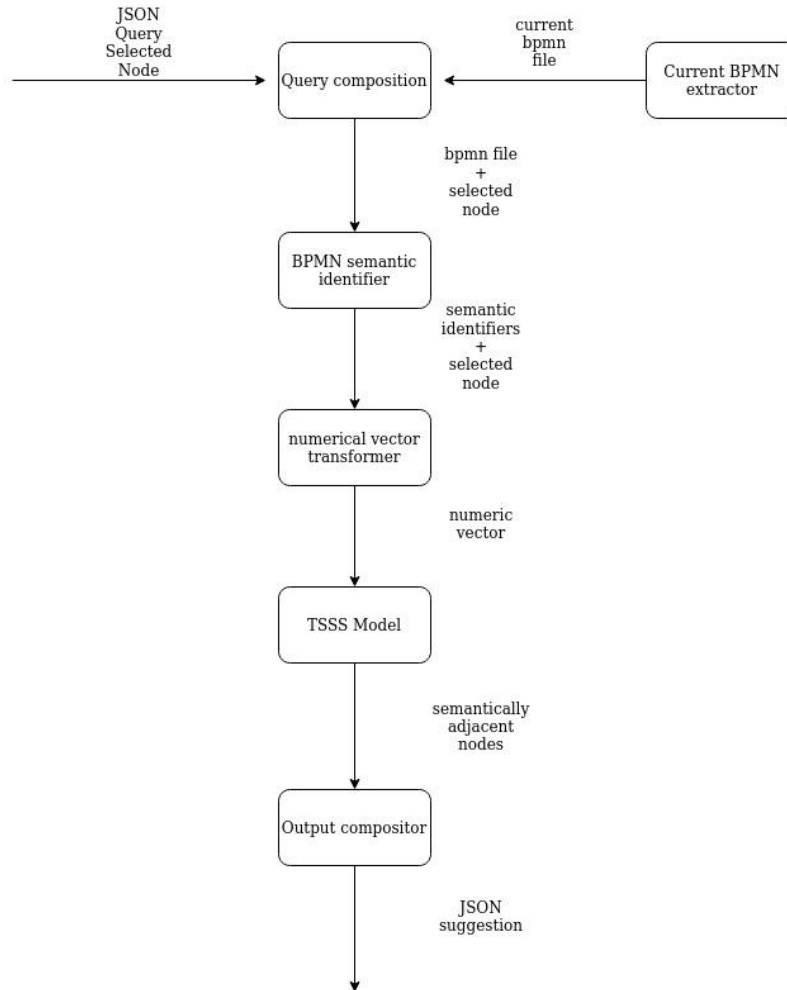


Figure 62: DLE BPMN Suggestions Architecture

Details of the stages are as follows:

- **Query Compositor.** Receives as input a JSON with the last selected node in the BPMN diagram and merges it with the incomplete bpmn file you are working with.

```

{
  "dle": {
    "header": "bpmn suggestion",
    "state": "query",
    "previous node": [
      {
        "id": "_13BAF867-3CA8-4C6F-85C6-D3FD748D07D2"
      },
      {
        "name": "UserFound?"
      }
    ]
  }
}
  
```

- **Current BPMN Extractor.** Query in user context the incomplete bpmn file that is being edited by the user. This information is obtained via REST from the Context Handling component through the SmartCLIDE MoM component.

- BPMN semantic identifier.** Based on network-based techniques to transform the incomplete BPMN into a text sequence.

At this stage, the aim is to simplify the BPMN XML which is composed of a large number of nodes and attributes that are only noise in this task. An example of a simplified XML approach is:

```

<bpnm2:definitions>
  <bpnm2:process id="users" name="users">
    <bpnm2:sequenceFlow id=" DBA10C00-6407-4EF5-9085-01177AE8F39F" sourceRef=" 5A1A031B-CA99-4CB7-BC07-A730CE950655" targetRef=" 08C87A94-E5F4-41B4-A388-3305342E9168">
    <bpnm2:sequenceFlow id=" BF17E37C-6984-4F27-986A-A9880E95B019" name="No" sourceRef=" 13BAF867-3CA8-4C6F-85C6-03FD748D07D2" targetRef=" 95885F94-555D-485A-BB86-5E835B9C3389">
    <bpnm2:sequenceFlow id=" 4EFC11AE-52BB-4EEF-B241-CFAAE4B7AE93" name="Yes" sourceRef=" 13BAF867-3CA8-4C6F-85C6-03FD748D07D2" targetRef=" E5D17755-D671-43ED-8D7D-F6538933069C">
    <bpnm2:sequenceFlow id=" 52A670E9-9448-48DA-8589-FC646BC41FC7" sourceRef=" 08C87A94-E5F4-41B4-A388-3305342E9168" targetRef=" 13BAF867-3CA8-4C6F-85C6-03FD748D07D2" />
    <bpnm2:sequenceFlow id=" 4EB288EA-3135-4B97-BB46-E77159F78832" sourceRef=" E5D17755-D671-43ED-8D7D-F6538933069C" targetRef=" FD4D7A19-558E-4347-8CFE-376792FEDA57">

    <bpnm2:startEvent id=" 5A1A031B-CA99-4CB7-BC07-A730CE950655" name="StartProcess">
    <bpnm2:outgoing> DBA10C00-6407-4EF5-9085-01177AE8F39F</bpnm2:outgoing>
    </bpnm2:startEvent>

    <bpnm2:serviceTask id=" 08C87A94-E5F4-41B4-A388-3305342E9168" name="Find user" implementation="Java" operationRef=" 08C87A94-E5F4-41B4-A388-3305342E9168_ServiceOperation">
    <bpnm2:incoming> DBA10C00-6407-4EF5-9085-01177AE8F39F</bpnm2:incoming>
    <bpnm2:outgoing> 52A670E9-9448-48DA-8589-FC646BC41FC7</bpnm2:outgoing>
    </bpnm2:serviceTask>

    <bpnm2:serviceTask id=" E5D17755-D671-43ED-8D7D-F6538933069C" name="Audit user" operationRef=" E5D17755-D671-43ED-8D7D-F6538933069C_ServiceOperation">
    <bpnm2:incoming> 4EFC11AE-52BB-4EEF-B241-CFAAE4B7AE93</bpnm2:incoming>
    <bpnm2:outgoing> 4EB288EA-3135-4B97-BB46-E77159F78832</bpnm2:outgoing>
    </bpnm2:serviceTask>

    <bpnm2:endEvent id=" 95885F94-555D-485A-BB86-5E835B9C3389" name="End Event 2">
    <bpnm2:incoming> BF17E37C-6984-4F27-986A-A9880E95B019</bpnm2:incoming>
    </bpnm2:endEvent>

    <bpnm2:exclusiveGateway id=" 13BAF867-3CA8-4C6F-85C6-03FD748D07D2" name="User found?" gatewayDirection="Diverging">
    <bpnm2:incoming> 52A670E9-9448-48DA-8589-FC646BC41FC7</bpnm2:incoming>
    <bpnm2:outgoing> 4EFC11AE-52BB-4EEF-B241-CFAAE4B7AE93</bpnm2:outgoing>
    <bpnm2:outgoing> BF17E37C-6984-4F27-986A-A9880E95B019</bpnm2:outgoing>
    </bpnm2:exclusiveGateway>

    <bpnm2:endEvent id=" FD4D7A19-558E-4347-8CFE-376792FEDA57" name="Done">
    <bpnm2:incoming> 4EB288EA-3135-4B97-BB46-E77159F78832</bpnm2:incoming>
    </bpnm2:endEvent>
  </bpnm2:process>
</bpnm2:definitions>
  
```

Figure 63: XML-BPMN simplified

In this way, the sequence of nodes involved can be extracted in a simpler way. From the BPMN file shown, we can obtain 2 linear sequences (all the combinations between start nodes and End nodes):

Find User - User Found? - Audit user - Done
Find User - User Found? - End Event 2

- Numerical vector transformer.** This stage is based on NLP (with the Word2Vec algorithm). The objective is to transform the incomplete BPMN into a numerical vector. We will work on an implementation with word embedding under Python with Spacy and Gensim. We will assess two lines of implementation. One based on the textual semantic identification of BPMNs under pre-trained models and the other one training directly with the corpus of services itself. Service similarity will initially be based on pre-trained NLP models so that it is the context of the user and the BPMN diagram that will determine the most recommended service. A further line of work would be to train the model with the services in the SmartCLIDE registry based on being represented in previous BPMNs. In this way, the similarity of services and therefore the recommendations will be based on whether a service has previously been linked to another in a BPMN diagram. The idea would be to look for the most similar services to the last ones included and suggest those.
- Transformed space similarity search (TSSS).** Based on similarity metrics defined in the transformed vector space created in the previous stage, tools are designed to locate the most suitable services for the BPMN design context in which the user is located. The last nodes introduced determine a textual pattern to which corresponds a representation in the transformed space that can be used for recommendation. This stage is at the heart of the success and failure of the recommendations. The aim is to suggest services from the internal SmartCLIDE registry during the composition of the workflow in BPMN, so as to avoid having to do a search to complete each node individually.

The model will be trained with all the services in the SmartCLIDE service registry so that similar services will be related. For this training we will use the total service registry transformed with the previously defined vectorisers. The homogeneity of the dataset will be analysed to determine its suitability and exploitability. For this purpose, clustering can be performed to see the uniformity of the groups that compose it. An example of a representation using a specialised technique to reduce the dimensionality of what could be obtained is shown in Figure 64.

When a suggestion request arrives, the previous stages extract in text form the semantics of the context that is represented jointly to the pre-trained model. In this way it can be represented in the current context on this model and thus obtain the numerical distance to the nearest services and collect them as output to the output composer stage.



Figure 64: DLE services dataset visual representation

Output compositor. This stage has several responsibilities:

- To improve the results obtained from the model by applying a system of penalties on the recommendation rating to services already present in the BPMN diagram to avoid making recommendations of nodes already present.
- Decide whether the results obtained from the model allow a recommendation to be made. The transformed vector similarity model results in the ordered list of services most similar to the context.
- Compose the recommendation in JSON format

```
{
  "dle": {
    "header": "bpmn suggestion",
    "state": "true",
    "previous node": [
      {
        "id": "_13BAF867-3CA8-4C6F-85C6-D3FD748D07D2"
      },
      {
        "name": "UserFound?"
      }
    ],
    "suggestionnode": [
      {
        "id": "_E5D17755-D671-43ED-BD7D-F6538933069C"
      },
      {
        "name": "AuditUser"
      }
    ]
  }
}
```



```
}
{
  "dle": {
    "header": "bpmnsuggestion",
    "state": "false",
    "previousnode": [
      {
        "id": "_13BAF867-3CA8-4C6F-85C6-D3FD748D07D2"
      },
      {
        "name": "UserFound?"
      }
    ]
  }
}
```

10 Cloud Application Testing

In this section, we describe the research approach taken to support the testing of the cloud applications created in SmartCLIDE.

10.1 Cloud Testing vs Testing a cloud

There is some confusion when it comes to testing in cloud environments, so some clarifications are needed. Testing a cloud refers to the validation and verification of the applications, environments, and infrastructure that are available on demand, ensuring that all of them conform to the expectations of the cloud computing business model. This means that for a cloud-based application, all of the traditional testing models must be applied for ensuring that all functional requirements are met and that the quality of the end product is high. But it also means that emphasis needs to be put on meeting non-functional requirements, as they are the base for the cloud computing business model.

Cloud Testing, on the other hand, is a type of software testing in which the software application is tested using cloud computing services. It is also called ‘Testing as a Service’, and its purpose is to perform the tests required to make sure that the cloud-based application meets its requirements, with a special focus on scalability, performance, security, and reliability. As the name indicates, this form of testing is performed on a third-party cloud computing environment that houses the required infrastructure to perform tests, which allows testing software and hardware without the usual constraints of budget, geographical location, number of test cases, costs per test, etc.

10.2 Types of Testing

Generally speaking, every software application development must involve several types of testing distributed along the lifecycle of the product. The purpose of all such testing is to ensure the product meets both functional and non-functional requirements, and so provide a high-quality end product that users will be happy to use. For cloud-based products, it’s essential to make sure that the product (or service) not only meets its functional requirements but also the non-functional requirements. So a strong emphasis needs to be laid on non-functional testing as well. The aforementioned testing types that a cloud-based application should have are described below in more detail.

10.2.1 Functional Testing

Functional Testing ensures that the product provides all the services and functionalities advised and that the business requirements are being met.

- **Unit Testing:** It is the kind of test performed by developers (who write the source code of the application) to achieve specific functionality for each unit of an application. During unit testing, it is made sure that each part of the application works as intended and provides the expected results.
- **Integration Testing:** It ensures that the modules of an application are working fine and don’t show any bugs when integrated. Integration tests allow operational commands and data to act as a whole system rather than individual components. It is generally performed to find issues with UI operations, operation timing, API calls, data formats, and database access.

- **Acceptance Testing:** These are those tests performed by a selected group of end-users, which will be given access to a functional version of the application and will eventually indicate whether the application is good enough (accepted) or not.

10.2.2 Non-Functional Testing

Non-Functional Testing focus on testing the cloud computing characteristics and features:

- **Availability Testing:** Cloud offerings must be available at all times. It is the responsibility of the Cloud vendor to ensure that there are no abrupt downtimes. In addition, the business of the client should not be adversely affected in case of any planned downtime
- **Security Testing:** There must be no unauthorized access to data. Shared data integrity should be maintained and secured at all times. At present several organizations and communities are formalizing industry standards to define the acceptability criteria for Cloud offering in terms of security
- **Performance Testing:** Performance measurement for a cloud service is different from an on-premise one. The Cloud should be elastic, in the sense that it enables enterprises to use limited resources from the Cloud application and increase the usage as required. Hence, the Cloud offering should be tested for fluctuating usage. The performance of the application should stay intact with maximum inflow of requests. Testing should also ensure that automatic deprovisioning occurs as the load decreases. To test the offering in case of increased load and stress, two of the following traditional performance testing methods are used:
 - Load testing
 - Stress testing
- **Interoperability Testing:** Any developed or migrated Cloud application must work in multiple environments and platforms. The application should also have the capability to be executed across various Cloud platforms. It should be easier to move the Cloud applications and platforms from one Infrastructure (as a service) to another Infrastructure. As with Security Testing, standards are being formalized for interoperability between diverse Cloud offerings too.
- **Disaster Recovery Testing:** It is preferred that a Cloud service is available all the time, though it is not 100% achievable even for on-premise applications. In case of a failure, the disaster recovery time must be low. Verification must be done to ensure the service is back online with minimum adverse effects on the client's business.
- **Multi-tenancy Testing:** Multi-tenancy refers to multiple clients and organizations using an on-demand offering. Considering the requirements to be verified for multi-tenancy, the service should be customizable for each client and it should provide data and configuration levels for security to avoid any access-related issues. A Cloud service should be thoroughly validated for each client whenever multiple clients are active at a given time.

In our case, we will focus mainly on Performance testing and how to automate it and integrate it into the development flow. A clarification needs to be done here since the term “performance” is a subject in other parts of this document (monitoring), as well. Performance monitoring is a complementary practice that gives an overview of the current state of the system, once it has been deployed. In contrast, the objective of performance testing, which is to ensure that the system will be capable of handling a required workload, and eventually provide data as to what is the limit of the system for handling higher workloads.

10.3 Automated Performance Testing

In Waterfall development, performance testing typically happens just before deployment. But if development and functional testing take longer than expected, there's less time for performance testing, which happens quite often. So, performance problems might be discovered too late, and worse, performance issues tend to be tricky to resolve, and in case of issues that are related to code or architecture, there is often no time to react.

Moving to an Agile development methodology and shifting performance testing closer to the development helps to resolve these issues earlier in the development cycle, when there is time to investigate and fix them.

For example, a microservices architecture is made up of small components. One of the advantages is that when there is an increased load or peaks, there is no need to re-start / redeploy the entire stack; one can just deploy more nodes. But this can create performance issues. If the application is consuming resources heavily, due to some bottleneck or resource leakage, it will keep spinning up more and more nodes. As a result, it might have delivered acceptable response times or user experience. But the cost of the infrastructure will have increased significantly. Performance testing can simulate this scenario before it happens.

Automated performance testing checks the speed, response time, reliability, resource usage, and scalability of software under an expected workload by leveraging automation. The goal of performance testing is to eliminate performance bottlenecks in the software.

There are many types of performance tests, each type serving a different purpose.

- Smoke Test's role is to verify that your System can handle minimal load, without any problems.
- Load Test is primarily concerned with assessing the performance of your system in terms of concurrent users or requests per second.
- Stress Test and Spike testing are concerned with assessing the limits of your system and stability under extreme conditions.
- Soak Test helps with the reliability and performance of a system over an extended period of time.

The important thing to understand is that each test can be performed with the same test script. A developer can write one script and perform all the aforementioned tests. The only thing that changes is the test configuration, the logic stays the same. There are two ways of creating these kinds of tests: API hammering or user flow scenarios.

API hammering refers to leveraging API specification documents, like OpenAPI/Swagger specifications. When such a specification exists, it is possible to automatically create tailored client applications that hit every public endpoint with samples of valid inputs. With some configuration, it is easy to increase the concurrent calls and generate load on those endpoints. However, these will be “dumb”, unrelated calls that might not expose real-world problems, which may occur during regular use of the application. These problems can only be exposed by scripting user flow scenarios.

User flow scenarios mimic a sequence of actions carried out by a real user. In this way, subsequent calls to different endpoints are related and can be correlated by the application. In this way, it is easy to explore what sequence of actions leads to a bottleneck or failure. Again, with some configuration, it is possible to simulate several users (up to thousands of them) using the application and measure the response of the system.

10.4 Integrating Performance Testing into Development Workflow

Given that most of the functional testing is already integrated into the development flow, and that most of the non-functional tests need a specific approach, the SmartCLIDE Cloud Testing will be to focus on integrating the Availability and Performance testing into the development cycle. In this way the user will be able to assess early on whether the application is meeting the requirements, instead of waiting until the end and having to do reactive programming to patch the source code to fix bottlenecks or underperforming code that causes failures when put under heavy load.

To integrate this kind of testing into the regular development flow, we propose to use a Behavior Driven Development approach (BDD) which allows people with little to no previous knowledge of coding, to write the specifications for the tests. By leveraging the right tools, we can generate the source code for the tests, which can be then integrated into the Continuous Integration (CI) engine, so that they can be run automatically, early on, and often, gaining all the benefits of the latest development trends: development efficiency, fast feedback, and improved software quality.

This approach assumes that SmartCLIDE will promote the use of BDD techniques, as initially stated in the product vision, providing IDE support for it.

10.5 Behaviour Driven Development

Behaviour Driven Development (BDD) is a synthesis and refinement of practices that evolved from Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD). BDD is also referred to as Specification by Example since requirements are written in a shared language, which improves communication between tech and non-tech teams and stakeholders.

BDD augments TDD and ATDD with the following tactics:

- Apply the “Five Why’s” principle to each proposed user story, so that its purpose is clearly related to business outcomes
- Thinking “from the outside in”, in other words, implement only those behaviors which contribute most directly to these business outcomes, so as to minimize waste
- Describe behaviors in a single notation that is directly accessible to domain experts, testers and developers, to improve communication
- Apply these techniques all the way down to the lowest levels of abstraction of the software, paying particular attention to the distribution of behavior, so that evolution remains cheap

BDD uses a clear structure to describe a situation, a flow of actions, and an expected result for them. The structure is simply written like:

- Given a certain situation
- When some specific user does something
- Then some result is generated

Gherkin is a common DSL (Domain Specific Language) used to implement BDD practices. It is highly structured, following the Given/When/Then pattern shown above, and it allows describing business behavior without going into details of implementation, but with enough detail to be parsed and generate a base skeleton for a set of tests.

D.2.1 SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment

Cucumber is a well-known framework capable of parsing Gherkin files and execute tests associated with the actions described in each scenario, technically resulting in Gherkin files being executed. Cucumber-js is an implementation of Cucumber functionality that works with JavaScript tests.

K6 is an open-source performance testing tool that is capable of executing test scenarios written in JavaScript. It is a flexible tool, whose goal is to have a developer-centric approach for the performance testing, seeking to integrate this form of testing into the development flow. As such, it seems like a perfect match for the approach.

11 Cloud Applications Security, Maintainability, and Reusability

In this section, we describe the methods for statically assessing the maintainability, reusability, and security of service-based software projects. In the first version of the deliverable (D.2.1) the consortium reports on methods and tools for assessing the quality at the project level, with traditional software engineering approaches. In the final version of the deliverable (D.2.2) the context will be enhanced with SOA-specific metrics and assessment methods.

11.1 Maintainability Assessment

11.1.1 Project-based Maintainability Assessment

Technical Debt. The most expensive activity in software development can be considered the maintenance, since it consumes 50 - 70% of the total effort spent by the developers in the lifecycle of a software [149]. In practice, some maintenance activities, for instance, the implementation of new functionality, the remediation of errors, the extension of supported environments, or the enhancement of run-time qualities, cannot be ignored or postponed. On the other hand, maintenance activities related to the improvement of design-time qualities, are usually postponed for future iterations, so as to reduce product time to market and diminish short-term costs. Nevertheless, software systems evolve profoundly over time and their quality will gradually decay [117], if developers neglect maintenance activities, such as refactorings, or resolution of bad smells. This can lead intentionally or unintentionally, to the creation of a financial overhead, that is now called Technical Debt.

Technical Debt (TD) is a metaphor borrowed from economics and it first appeared in the literature of software engineering in 1992 by Ward Cunningham. Cunningham used this metaphor in order to explain the risks of delivering first-time code with quick fixes, introducing the “debt” and “interest” in software engineering [39]. In the last decade, the TD metaphor became a very famous term both in the research and in the industry community, with the main reason being the connection and the communication channel that can be offered between the developers and the stakeholders of a project [84].

The two main concepts of TD are “principal” and “interest”, which have been tailored from economics science to software engineering. The next Figure presents the relationship between those two concepts according to Chatzigeorgiou et al. [31]. In this figure for every system state, the y axis represents the quality of the system and it is clear the distance between the actual and the “Optimum” system quality. The effort required by the development team to achieve the optimum quality represents the TD Principal. TD Interest is the negative effect and it represents the extra effort that is required for the maintenance of the system in the actual state, in comparison with the optimum state.

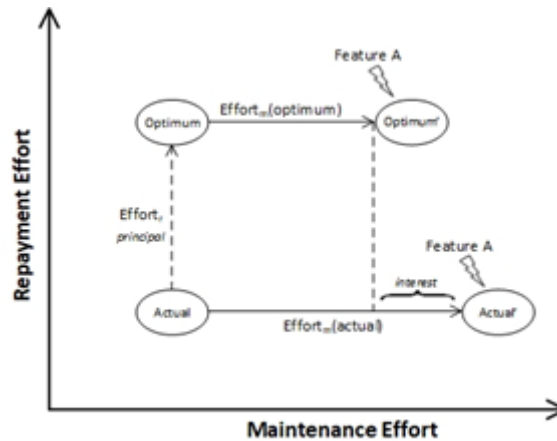


Figure 65: TD Principal and Interest [31]

For the calculation of TD Principal multiple tools have been introduced that measure the effort that is required to fix inefficiencies in the codebase. On the other hand, TD Interest is harder to be estimated because of the hypothetical optimum version of the system. Even though TD Interest can not be quantified with accuracy, we can find multiple ways that have been proposed for its estimation. Conejero et al. [38] noted that “maintainability is one of the main characteristics contributing to Technical Debt interest”. Moreover, [168] propose the use of defect and change proneness as proxies for TD Interest, as they acknowledge their connection to the future maintenance cost. Similarly, McCormack and Sturtevant [97] analyzed the relationship between design decisions and maintenance costs, considered to represent TD Interest, while Kazman et al. [78] focus on the relationship between architecture roots, like defective structures, and their consequences in terms of higher maintenance costs. Under this assumption, metrics that assess maintainability by quantifying design-time characteristics, such as inheritance, cohesion, coupling, complexity, and size, can be used as proxies for TD Interest estimation. However, the actual quantification of the TD Interest still remains open for research.

For the wellbeing of the software development company, TD should be managed, or else the accumulation of TD will become so large that could eventually bankrupt the company [11]. According to Li et al. [89] the management of TD consists of the following eight activities:

1. repayment (i.e., reducing the amount of accumulated TD)
2. identification (i.e., finding the artifacts that suffer from excessive TD values)
3. measurement (i.e., quantifying TD principal and interest)
4. monitoring (i.e., recording and assessing the evolution of TD)
5. prioritization (i.e., identifying the TD items that have to be repaid first)
6. communication (i.e., explaining the roots and consequences of TD to stakeholders)
7. prevention (i.e., avoiding the accumulation of additional TD in the first place)
8. representation / documentation (i.e., recording all decision, actions, metrics, related to TD)

As noted by [47], the full repayment of TD is considered unrealistic. For this reason, Technical Debt should be constantly managed, with the aforementioned activities, because it can really damage all the development processes [84]. From all types of TD, code TD is reported as the most frequently studied type in research [5] and a very crucial type in the industry [6].

Quantification Approach. The most frequently used tool for the calculation of TD Principal, according to Yli-Huumo et al. [162], is considered to be SonarQube. SonarQube quantifies the TD Principal through

the number of code inefficiencies according to a set of rules, and the amount of time required to fix these inefficiencies. Finally, the algorithm calculates the TD as the sum of remediation cost for all quality issues. Due to the high usage, extended language support, and the fact that is being already used by the pilot Intrasoft, our initial approach for the quantification of the TD Principal relies on SonarQube.

SonarQube can be used in a variety of ways, one of the easiest ways is to download the SonarQube instance and run it locally or download a SonarQube image and run it with Docker Desktop. In both cases when the SonarQube is up and running, a browser should be opened with the URL of “http://localhost:9000” in order to have access to the main dashboard. Port 9000 is the default port used by SonarQube, but if someone wants it could be replaced. Moreover, for the execution of the analysis, a separate tool is required, named SonarScanner. SonarScanner can be used in conjunction with automated build tools (i.e. maven, gradle, ant) or it can be used as a standalone tool.

In our case, we are going to use SonarQube's integration with GitLab, because we already selected GitLab as our main Git repository for the creation of services. Hence, the developers will be able to a) sign in SonarQube with your GitLab credentials, b) import GitLab projects into SonarQube to easily set up SonarQube projects, and c) integrate analysis into the build pipeline. In order to use the autonomous analysis in the build pipeline of a service, two main steps need to have taken place beforehand.

First of all, GitLab needs to be connected with the SonarQube instance, which is required only once for all the GitLab projects. The connection takes place by setting a SonarQube token (SONAR_TOKEN) and URL (SONAR_HOST_URL) in the environment variables for all pipelines in GitLab's settings. Second, the GitLab CI/CD needs to be configured, in order to add the sonar analysis in the pipeline. To configure the GitLab CI/CD a “gitlab-ci.yml” file needs to be created in the root directory of the project. The content of this file is strictly connected to the automated build tool that it’s being used. In the following Figure is presented the file with maven as the automated build tool. Finally, if the Jenkins pipeline is being used, the SonarQube documentation needs to be addressed accordingly for the appropriate configuration. In the case of Jenkins a plugin for the connection with GitLab is required along with an extra stage in the “Jenkinsfile” file for the pipeline.

```
sonarqube-check:
  image: maven:3.6.3-jdk-11
  variables:
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar" # Defines the location of the analysis
task cache
  GIT_DEPTH: "0" # Tells git to fetch all the branches of the project, required by the analysis task
cache:
  key: "${CI_JOB_NAME}"
  paths:
    - .sonar/cache
script:
  - mvn verify sonar:sonar -Dsonar.qualitygate.wait=true
allow_failure: true
only:
  - master
```

Figure 66: Contents of gitlab-ci.yml file for SonarQube analysis

On the other hand, regarding the calculation of TD Interest we emphasise on maintainability, as a proxy to the amount of TD interest. Because in object-oriented programming languages maintainability has been heavily assessed by Riaz [127] we use a selected number of metrics for our calculations. First, in order to find the five most similar files of each file, and second to find the difference of the metric's values of each file with most similar and optimal one. Finally, we can get the interest in euros by multiplying with the change proneness of that file along with the payment of the developers per hour.

11.1.2 Service-based Maintainability Assessment

This section will be completed in the final version of this deliverable, i.e., deliverable D.2.2

11.2 Reusability Assessment

11.2.1 Project-based Reusability Assessment

Washizaki et al. [153] proposed a metric suite, consisting of six software quality metrics, capturing the reusability of black-boxed components. The validity of the model was evaluated with 125 components against expert estimations. Bansiya and Davis [15] proposed a hierarchical quality model, named QMOOD, for surveying the quality of object-oriented artifacts and depended on human evaluators to survey its legitimacy. This quality model provides methods that relate structural properties (i.e., coupling and cohesion) to high-level quality attributes, among of which is reusability. Moreover, [108] look at the reusability of a certain class, based on the values of three measurements characterized within the Chidamber and Kemerer [34] metrics suite. Multifunctional relapse was performed over measurements to characterize the Reusability List which was assessed in two medium-sized java ventures. Also, Kakarontzas et al. [75] proposed an index for surveying the reuse potential of object-oriented computer program modules. The creators utilized the measurements presented by Chidamber and Kemerer [34] and derived a new reusability index (named FWBR) based on the results of a linear regression performed on 29 OSS ventures. [8] propose a reusability index, named REI, an index that takes into consideration reusability at different perspectives: such as structural quality, external quality, documentation, availability, etc. The index proposed is calculated by taking into consideration seven metrics that correspond to both structural and non-structural quality characteristics. To validate the accuracy of the developed index, a case study containing 15 well-known open-source assets was performed.

From another point of view, [138] made use of Artificial Neural Networks (AAN) to export an estimation of the overall reusability of software components. The logical basis of their model is that structural software quality metrics cannot be the only indicators of components reusability, due to the fact that reusability can be performed in manifold levels of granularity. The authors developed a network consisting of 40 Java components and tested their results in 12 components.

To measure the reusability index in our case, we will use the generalized formula derived from the publication of Kakarontzas et al [75]. The exact formula is:

$$\text{FWBR} = -1 \times (8.753 \times \log(\text{CBO} + 1) + 2.505 \times \log(\text{DIT} + 1) - 1.922 \times \log(\text{WMC} + 1) + 0.892 \times \log(\text{RFC} + 1) - 0.399 \times \log(\text{LCOM} + 1) - 1.080 \times \log(\text{NOC} + 1))$$

Where:

- CBO: Coupling Between Objects

- DIT: Depth of Inheritance
- WMC: Weighted Methods per Class
- RFC: Response for a Class
- LCOM: Lack of Cohesion Among Methods
- NOC: Number of Children

To calculate the metrics used by the above formula, we use a static code analysis tool, which extracts many code metrics from various software quality metrics packages, such as Chidamber and Kemerer [34], Li and Henry [90], Bansyia [15].

11.2.2 Service-based Reusability Assessment

This section will be completed in the final version of this deliverable, i.e., deliverable D.2.2

11.3 Security Assessment

11.3.1 Vulnerability Assessment

Software security is an aspect of major importance for software development enterprises that wish to deliver secure and reliable software to their customers. Modern software applications are usually interconnected through the internet and handle sensitive information. Therefore they are constantly at risk of malicious attacks. The exploitation of a single vulnerability may lead to far reaching consequences both for the end user (e.g., information leakage) and for the owning enterprise of the compromised software (e.g., financial losses and reputation damages) [58]. As a result, the software industry has turned its attention towards developing techniques that can provide indicative information to developers about the security level of their software by identifying vulnerable hot-spots in the source code.

One such mechanism that enables the prediction and mitigation of software vulnerabilities early enough in the development cycle is the Vulnerability Prediction (VP). VP models (VPM) can be used for prioritizing testing and inspection efforts, by allocating limited test resources to potentially vulnerable parts. Several VPMs have been proposed over the years using various software factors as inputs including software metrics, static analysis alerts and a text mining technique named bag of words (BoW). Although these models have demonstrated promising results, there is a need for improvement. Static analysis alerts include a lot of false positives among with the serious alerts. BoW method seems to provide better results than the static analysis alerts and the use of software metrics but it is too dependent on the software project that is used for the model training. Therefore, recent studies have shifted their focus to more sophisticated methods that can detect patterns in source code that indicate to the existence of a vulnerability. They focus either on extracting information from the raw source code of a given software application, or from abstract representations of their source code, such as their Abstract Syntax Tree.

This study develops deep-learning (DL) models capable of predicting whether a software component is vulnerable, using the raw text of the source code in the form of sequences of instructions, utilizing methods from the field of natural language processing (NLP) and text classification. For this purpose, we adopted techniques from the NLP field. Source code is considered as text and the vulnerability assessment task is considered as a text classification task such as the sentiment analysis. So by applying NLP techniques such as the Bidirectional Encoder Representations from Transformers (BERT) [46], data pre-processing and transformation to sequences and by training DL models (e.g. recurrent neural

networks) suitable for analyzing sequential data, we detect potential vulnerable components through a binary classifier, which is trained mainly on the source code's sequences of text tokens. Furthermore, software metrics collected by static code analyzers could be also used, along with text mining techniques, in order to enhance the predictive performance of the models.

Related Work. Vulnerability assessment models are normally built based on machine learning techniques that use software metrics as input, to discriminate software components between vulnerable and clean components. Two of the most widely studied factors are software metrics and text features retrieved through text mining. Hence, we can split these models in two main categories, the models that use software metrics [170] and models that use text mining [115] [136]. Text mining models have demonstrate the most promising results so far [42], [65] [136].

Text mining approaches parse the source code of software components and represent it as a set of code-tokens, which are then used to train predictors. Vulture [109] was the first framework proposed, a VPM that predicted vulnerabilities based on import statements and function calls that are more commonly found in vulnerable components. Vulture was tested on Mozilla Firefox and Thunderbird code, and the results were promising. Hovsepyan et al. [65] presented a more comprehensive text mining-based prediction approach. They parsed the source code of software components to extract text terms and their frequency distributions to use as features in their predictors. That method is known as the Bag of Words (BoW) method. An empirical evaluation of their technique on 19 versions of a large-scale Android application revealed that it may be promising for vulnerability prediction, as the produced predictors achieved sufficient precision (85 percent on average) and recall (87 percent on average). The same authors expand their research by feeding code-tokens and their frequencies into Nave Bayes and Random Forest predictors on a dataset of 20 different Android apps [136]. Their findings suggest that text mining is capable of producing adequate models for identifying vulnerable classes. The BoW approach, on the other hand, fails to recognize semantic relationships between words, and a bag of code tokens does not always capture the syntactic structure of code, particularly its sequential nature.

Instead of using raw text features, Pang et al. [115] used N-Gram analysis¹ to represent source code as continuous sequences of tokens. Actually, they combined N-gram analysis and statistical feature selection to create features, and they used a deep neural network to predict vulnerable software components, testing their findings on a variety of Java Android programs. According to the evaluation results, the approach can provide satisfactory suggestions with high precision, accuracy, and recall. However, because the evaluation was based on a small dataset, additional analysis would be required to ensure the generalizability of the findings.

Dam et al. took a more complicated approach [41]. They concentrated on capturing source code's semantic and syntactic representations. They built a deep learning Long Short Term Memory model (LSTM) for feature extraction and tried Random Forest, Decision Tree, Naive Bayes, and Logistic Regression as classifiers to automatically learn both semantic and syntactic features of code. Their tests on 18 Android apps and the Firefox application yielded better results than previous state-of-the-art approaches. Despite these promising results, their model is based on a model trained to predict the next token in source code [42], which makes it computationally expensive, necessitating a large memory space and lengthy training time. Furthermore, rather than using any Natural Language Processing (NLP) technique, they relied solely on the LSTM model's power to capture syntactic information. To detect a variety of vulnerabilities, Russel et al. [134] used deep learning to learn features directly from source code in a large natural Cpp codebase. They trained a CNN to extract features from source code, and then

fed the extracted features to a Random Forest classifier to classify functions as vulnerable or clean. Despite experimenting with both 1-hot encoding and word2vec approaches, they were unable to improve their classification performance. As a result, they used small custom embeddings and added random Gaussian noise to the embedding vectors.

Li et al. presented a deep learning model for vulnerability detection in their study, VulDeePecker [91]. They divided the source code into a number of lines of code that are semantically related to one another, and then used the word2vec tool to convert them into vectors. They developed a Bidirectional LSTM (BLSTM) model to detect library/API function calls associated with known vulnerabilities. Although the authors concentrated solely on vulnerability detection, their concept could have been expanded to investigate its potential in vulnerability prediction.

Walden et al. [152] compared text mining-based vulnerability prediction models to models that used software metrics as predictors. They provided a dataset containing 223 vulnerabilities discovered in three web applications for this purpose (Drupal, Moodle, PHPMyAdmin). Random Forest models were trained in this study to predict vulnerable and clean PHP files. The results show that text mining outperforms software metrics for project-specific vulnerability prediction, but falls short for cross-project prediction, where software metrics perform better. Approaches to combining software metrics and text mining techniques, on the other hand, have been investigated. VULPREDICTOR [171] is a method proposed by Zhang et al. that employs an ensemble of classifiers with a combination of text and software metrics as features. The evaluation results suggest that combining software metrics and text mining for vulnerability prediction may be promising, as they outperformed the results produced by Walden et al. [152], who used software metrics or text mining separately.

Based on the Abstract Syntax Trees (AST) representation of source code, Bilgin et al. [20] created an ML model for vulnerability prediction. They concentrated on AST encoding to numerical tuples. Cao et al. [28] proposed using a bidirectional Graph Neural Network (GNN) to detect vulnerabilities. To capture syntactic and semantic information, they used not only AST but also Control Flow Graph (CFG) and Data Flow Graph (DFG). They discovered that GNN can handle non-sequential features of graph-based data better than LSTMs or CNN. They used the embedding model word2vec for word representation and trained a CNN for classification. Devign, a general graph neural network-based model, was proposed by Zhou et al. [173]. Devign included a novel "Conv" module for efficiently extracting useful features from learned rich node representations for graph-level classification. They also concluded that combining AST, CFG, DFG, and natural code sequences yielded better results than using each one separately.

To sum up, based on the aforementioned literature, one can identify that there is a lot of room for working on more advanced approaches in text mining-based vulnerability prediction methods. There are sophisticated NLP techniques that can be adopted to detect vulnerabilities. BERT is a famous pre-trained model that can be fine-tuned for a specific task, such as the vulnerability assessment that uses the raw source code as input. In the field of software security, Yin et al. applied transfer learning to predict the exploitability of the vulnerabilities based on the vulnerability description corpus [161]. They used the pre-trained BERT model in order to gain a better performance since the size of vulnerability description corpus is too small. Finally, the combination of software metrics and text mining techniques is also a topic that has not been explored thoroughly. There are no advanced deep learning models proposed that can combine the text tokens with the knowledge extracted from software metrics and probable we could explore this field to ascertain if it can be proved beneficial for the predictive performance of VPMs.

Theoretical Background. In this section we present the theoretical background of the technologies that we use. The information provided in this section is very important for familiarizing the reader with the main concepts of the present work.

Vulnerability Assessment is actually the process of highlighting hot-spots in the software that are more likely to contain software vulnerabilities. These hot-spots are considered as security issues that need to be fixed by the developers. The VPMs provide to the developers a list of all the identifying security hot spots, i.e., software components that are likely to contain vulnerabilities. Among these alerts there are also some false positives that increase the amount of time the developers have to spend in order to fix the vulnerabilities. VP research aims to optimize models so as to provide as less false alerts as possible and as many true vulnerabilities as possible. VPMs usually output files with all the analyzed software components accompanied with a vulnerability flag that declares the existence or the absence of vulnerabilities to a specific component (i.e. class, function) and they often include also a score representing the likelihood of a component to be vulnerable. These models are mainly machine learning (ML) and deep learning (DL) models that use software indicators as features; either statically extracted software metrics [139] [140] [141] or text features from the source code [65], [109]. In case of using text features, these models are called text mining models and are the most promising ones, having attracted the most research focus recently [41], [115], [134].

Text Mining is the process of tokenizing a text such as the source code. BoW constitutes the simplest text mining method [136]. In BoW, the code is splitted to text tokens each one of them is accompanied by the number of its appearances in the code. So each word corresponds to a feature and its frequency in a component makes up the value of this feature for this specific component. Apart from BoW, text mining adopts methodologies from the field of natural language processing like using word2vec pre-trained embedding vectors to capture semantic information from the tokens [91]. Text mining also includes the process of transforming the source code to a list of sequences of tokens in order to use these sequences as input to DL models capable of parsing sequential data (e.g., recurrent neural networks, BERT) [41]. Text mining along with DL can enhance the performance of the VPMs significantly and has attracted the research interest recently [41], [115], [134].

Bidirectional Encoder Representations from Transformers. The majority of cutting-edge NLP systems relied on gated recurrent neural networks (RNNs), such as long short-term memory networks (LSTMs) and gated recurrent units (GRUs), which are typically used as both encoder and decoder. Bahdanau et al. [13] introduced a mechanism called attention. Deep Learning's Attention mechanism is based on the concept of directing the focus and paying more attention to certain factors when processing data, dealing with the problem of long sequences. In 2017, Vaswan et al. published the paper “Attention is all you need” [150], introducing the Transformer mechanism. The Transformer was built without using an RNN structure, demonstrating that attention alone, without recurrent sequential processing, is powerful enough to achieve the performance of RNNs with attention while being more parallelizable and requiring significantly less time to train.

The RNN-based sequence-to-sequence (e.g. machine translation, speech recognition) model consists of an encoder and a decoder both of which containing RNN layers as can be seen in the Figure 67:

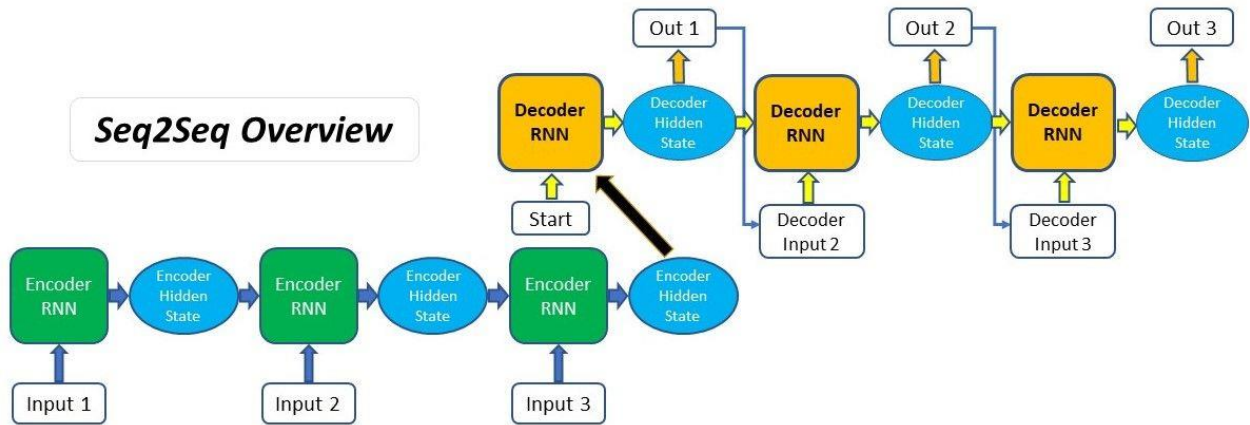


Figure 67: Sequence to Sequence model Overview

The input is parsed through the recurrent layers of the encoder and the last hidden state is passed to the decoder. Then the output is passed to the decoder which is trained on predicting the next word of the output sequence. Attention (that is illustrated in Figure 68) is a mechanism that manages to overpass the limitations of the different sequence length between input and output and also to direct model’s focus on the right part of the sequence, as can be seen in Figure 69.

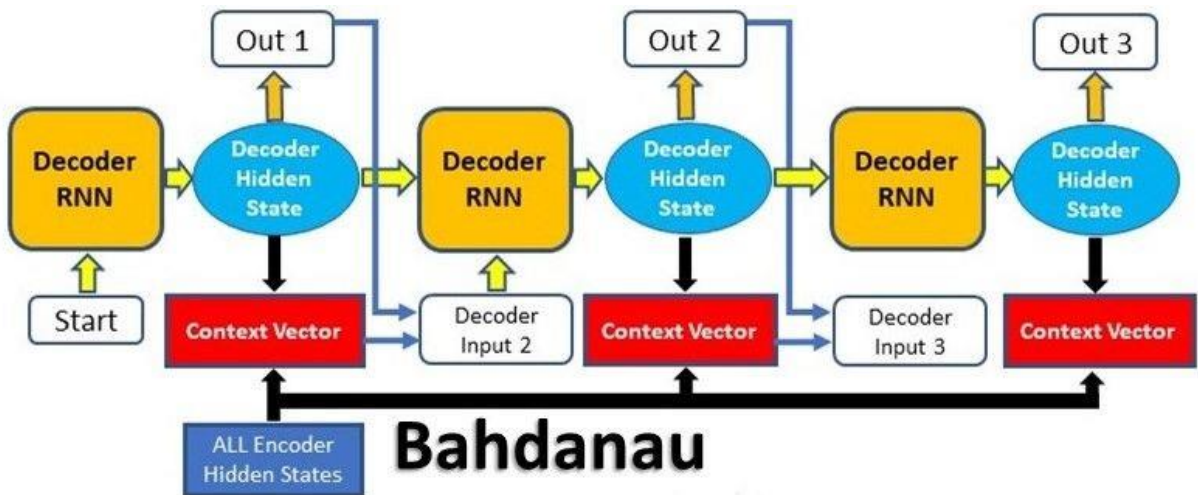


Figure 68: The Attention overview

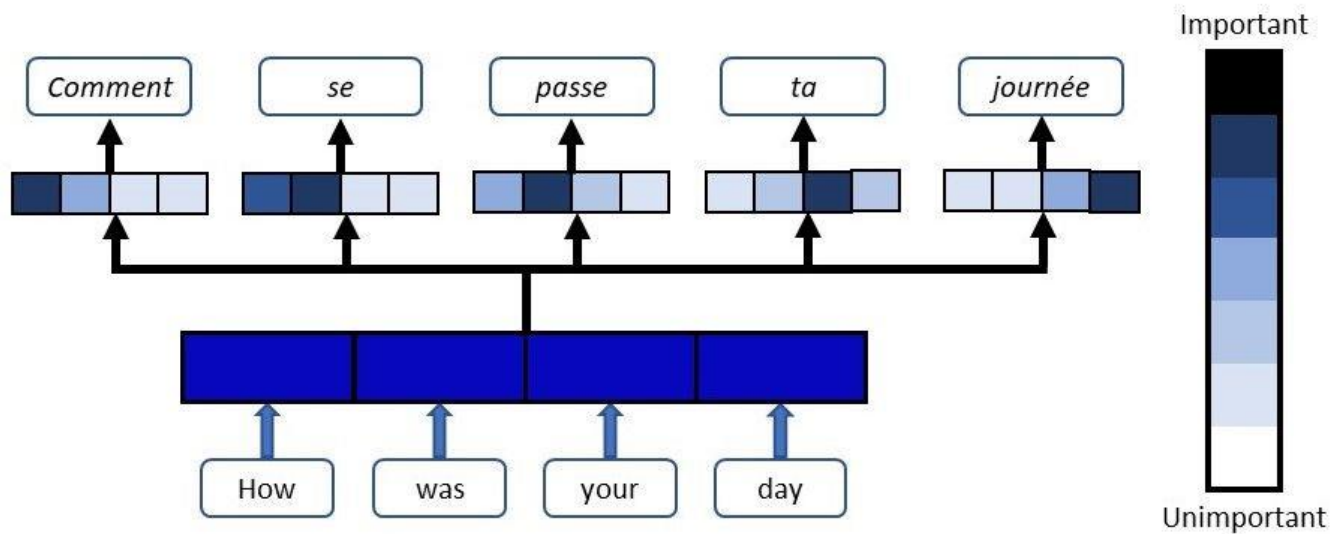


Figure 69: An example of Attention usage

The Transformer model is a more advanced step in the process. A machine translation application would take a sentence in one language and translate it into another. We can see in the Figure 70 an encoding component, a decoding component, and connections between them on the inside. The encoding component is made up of $N = 6$ identical layers. The decoding component is a stack of identical decoders.

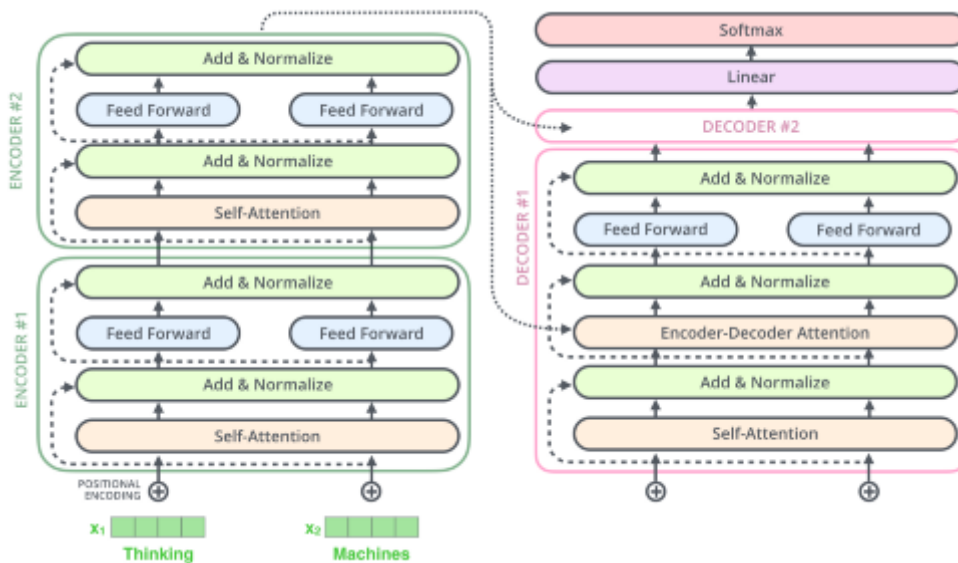


Figure 70: Transformer overall architecture

Using an embedding algorithm, we convert each input word into a vector, as is common in NLP. The encoder's inputs first pass through a self-attention layer, which assists the encoder in looking at other words in the input sentence while encoding a specific word. The self-attention layer's outputs are fed into a feed-forward neural network. The same feed-forward network is applied to each position independently. Both of these layers are present in the decoder, but in between them is an attention layer that assists the decoder in focusing on relevant parts of the input sentence.

BERT makes use of Transformer. In its vanilla form, Transformer consists of two separate mechanisms: an encoder that reads the text input and a decoder that generates a prediction for the task. Because the goal of BERT is to generate a language model, only the encoder mechanism is required. BERT is basically a trained Transformer Encoder stack [46]. The Transformer encoder reads the entire sequence of words at once, as opposed to directional models, which read the text input sequentially (left-to-right or right-to-left). As a result, it is regarded as bidirectional, though it would be more accurate to describe it as non-directional. This feature enables the model to learn the context of a word based on its surroundings (left and right of the word). The Transformer encoder is described in detail in the Figure 71. The input is a series of tokens that are embedded into vectors before being processed by the neural network. The output is a sequence of H-dimensional vectors, each vector corresponding to an input token with the same index.

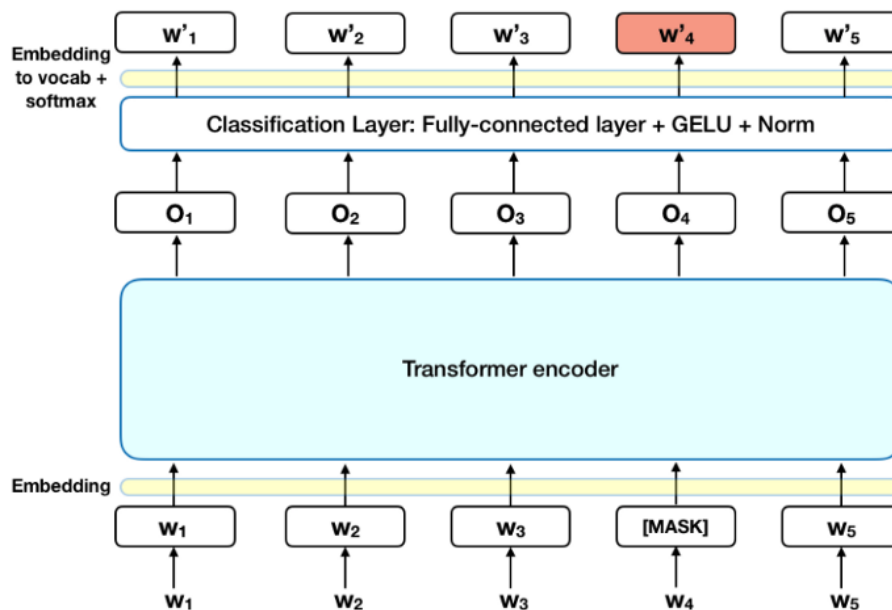


Figure 71: The Transformer Encoder model (BERT)

It is difficult to define a prediction goal when training language models. Many models predict the next word in a sequence, a directional approach that limits context learning inherently. BERT employs two training strategies to overcome this obstacle:

Masked LM (MLM). Before feeding word sequences into BERT, 15% of each sequence is replaced with a [MASK] token. Based on the context provided by the other, non-masked words in the sequence, the model then attempts to predict the original value of the masked words. In technical terms, the output word prediction necessitates:

1. On top of the encoder output, a classification layer is added.
2. The output vectors are transformed into the vocabulary dimension by multiplying them by the embedding matrix.
3. Softmax is used to calculate the probability of each word in the vocabulary.

The BERT loss function considers only the prediction of masked values and disregards the prediction of non-masked words. As a result, the model converges slower than directional models, but this is offset by its increased context awareness.

- Next Sentence Prediction (NSP)

During the BERT training process, the model is fed pairs of sentences and learns to predict whether the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are pairs in which the second sentence is the following sentence in the original document, while the other 50% are random sentences from the corpus. The random sentence is assumed to be disconnected from the first sentence. To help the model in distinguishing between the two sentences during training, the input is processed as follows before entering the model:

- A [CLS] token is placed at the start of the first sentence, and a [SEP] token is placed at the end of each sentence.
- Each token receives a sentence embedding indicating Sentence A or Sentence B. Sentence embeddings are conceptually similar to token embeddings with a vocabulary of 2.
- Each token is given a positional embedding to indicate its position in the sequence.

The following steps are followed to predict whether or not the second sentence is related to the first:

1. The Transformer model is received the entire input sequence.
2. Using a simple classification layer, the [CLS] token output is transformed into a 2x1 shaped vector (learned matrices of weights and biases).
3. Make use of softmax to calculate the probability of IsNextSequence.

Masked LM and Next Sentence Prediction are trained together in the BERT model, with the goal of minimizing the combined loss function of the two strategies.

BERT can be used for various NLP tasks (e.g. text classification, sentiment analysis) by applying fine-tuning to a pre-trained model. We simply plug in the task-specific inputs and outputs into BERT and fine-tune all the parameters end-to-end for each task. Fine-tuning is relatively inexpensive when compared to pre-training.

11.3.2 Methodology

In this section, we describe the whole methodology that we followed in order to design our vulnerability prediction models. The method that we propose is based on utilizing the sequential form of the source code instructions in the right order and trying to identify code patterns that are indicative for the existence of vulnerabilities in the source code. This technique is a text mining technique, but different from the more simplistic BoW method that is usually met in the bibliography. The software component level that is chosen for the vulnerability analysis is the class level.

The methodology is based on the utilization of the pre-trained BERT model, which is described thoroughly in the previous subsection. We fine-tuned this model to adjust it to the classification task of vulnerability assessment. As input of the model are considered the sequences of text tokens of the class files, while the output is a vulnerability flag that indicates the existence of vulnerabilities in a specific file.

11.3.3 Datasets

As part of the present work, we built two different VPMs, one for two widely-used programming languages, namely C/C++ and Java. For the case of C/C++, we utilized a vulnerability dataset derived from two National Institute of Standards and Technology (NIST) data sources: the National Vulnerability Database (NVD) and the Software Assurance Reference Dataset (SARD). This dataset consists of 7651 class files, 3438 of which are considered as vulnerable and the rest 4213 are considered as clean.

In case of the Java model, a dataset provided by OWASP3 is utilized in order to train and evaluate the produced model. This dataset contains 1415 vulnerable class files and 1325 class files considered as clean.

11.3.4 Pre-processing

We collected source code files written both in Java and Cpp programming languages and we applied a series of pre-processing techniques in order to transform the datasets in a sequence of words-tokens. All comments were removed from the dataset as well as the header/import instructions that declare the use of specific libraries in the class. Then the numeric values (i.e. integers, floats etc.) were replaced by a unique identifier “numId\$”, while the strings values and the characters were replaced by another unique identifier “stride\$”. All the blank lines are also removed and finally the text is transformed to a list of code tokens (i.e. new, char, strlen etc.) in the order that they appear in the source code.

After data cleansing, these produced tokens are replaced by an integer (integer encoding⁴) because ML models understand numerical values. These integers are then correspond to vectors that are called embedding vectors. These vectors have a specific size and are numerical representations of the text tokens. Embeddings are important as they can represent a token better than a simple integer values.

11.3.5 Performance Metrics

There are several performance indicators available in the literature that are commonly used to evaluate the predictive performance of ML models. These performance indicators are typically calculated based on the number of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) produced by the models. In the case of vulnerability assessment, similarly to previous works, we place a special emphasis on the Recall (R) of the generated models, because the higher the Recall of the model, the more actual vulnerabilities it predicts. Aside from the ability of the produced models to accurately detect the vast majority of vulnerable files contained in a software product, it is important to consider the volume of the produced FP, (i.e., neutral files marked as vulnerable by the models), because it is known to affect the models' practicability. A large number of FP forces developers to inspect a non-trivial number of non-vulnerable files in order to detect a vulnerable file. As a result, the number of FP is proportional to the manual effort required by developers to identify files that actually contain vulnerabilities. The higher the precision is the lowest the number of FP is. So we have to take care of both recall and precision. This fact highlights the importance of the f1-score which conveys the balance of precision and recall. However, because in VP is more important to identify the vulnerable files in the expense of producing FP (but not too many) we choose f2-score as our evaluation metric in order to tune our models and evaluate them in the testing datasets. The f2-score is a weighted average of precision and recall, where the recall has a higher weight than precision. It is equal to:

$$F2 = (1)$$

11.3.6 Results

The model that is used is a pre-trained BERT model. Actually it is the BERT for sequence classification pre-trained model. It belongs to the category of BERT base [46] models as regarding its size. This means that the model has the following characteristics:

Table 14: Hyper-parameters of BERT base model

Parameters	
Number of layers	12
Hidden size	768
Total parameters	110M

After fine-tuning to our dataset, we ended up with these hyper-parameters:

Table 15: Fine-tuning hyper-parameters

Parameters	
Learning rate	2e-5
Number of epochs	2-4
Batch size	2

As regard the number of epochs, for the Java model was selected as two whereas for the Cpp model we needed four epochs. The batch size is just two batches because of limitations in the hardware used for training. Actually, the models were trained in Cuda (version 11.3), which is the parallel computing platform and application programming interface model created by Nvidia. From a hardware perspective, the training process took place in an Nvidia graphs processing unit (GPU) with memory equal to 6 gigabytes. The fine-tuning and the training were implemented in pyTorch-gpu5framework using the python programming language.

The vulnerability assessment evaluation was implemented both for OWASP Java dataset and also for the Cpp dataset. The evaluation algorithm is the 10-fold cross-validation in order to have the most generalized results. The results are summarized in the table below:

Table 16: Evaluation results

Model	F2 score (%)
Java	70.01
Cpp	78.73

The results of the experiments indicate that the models can identify vulnerabilities in the software in a satisfying degree. However, in the above table, one could easily notice that the model for Java has lower f2 score than that for Cpp. The reason why it is happening could be either the structure of the datasets or the fact that the OWASP dataset is much smaller than the Cpp dataset and so there are not enough Java files to analyze in order to reach an f2score equal to almost 79% as in Cpp case. Further investigation will be held as part of SmartCLIDE in order to lead to more accurate vulnerability prediction models. In fact, the use of BERT models in the field of vulnerability assessment has not been studied in the related literature so far, which showcases the novelty of the present work. In addition, F2 score between 70 and 79 percent is a promising performance as a first examination of the BERT utilization in VP, but we have to proceed to more experiments to improve this performance.

Below, two case studies are provided, one for Cpp and one for Java project. The primary goal of these case studies is to help the reader in understanding how the proposed solution works and what benefits can be expected.

For the purpose of the Cpp case study, the project in the GitHub repository with the url <https://github.com/akshitagit/CPPIs> is used and we provide the output of the model in JSON form, for a fragment of the whole analyzed project:

```
[
  {
    "file_path": "testRepo/CPP\\AdHoc\\Bit Hacks.cpp",
    "vulnerability_score": 0.0062083495,
    "vulnerability_flag": 0
  },
  {
    "file_path": "testRepo/CPP\\AdHoc\\FastIO.cpp",
    "vulnerability_score": 0.8422008157,
    "vulnerability_flag": 1
  },
  {
    "file_path": "testRepo/CPP\\AdHoc\\Gray Code.cpp",
    "vulnerability_score": 0.9539647698,
    "vulnerability_flag": 1
  },
  {
    "file_path": "testRepo/CPP\\AdHoc\\Infinite Grid.cpp",
    "vulnerability_score": 0.0024059685,
    "vulnerability_flag": 0
  },
  {
    "file_path": "testRepo/CPP\\AdHoc\\Josephus Problem.cpp",
    "vulnerability_score": 0.0110375136,
    "vulnerability_flag": 0
  }
]
```


]

Figure 72: Model output (JSON format) for the Cpp case study

The same process is followed for the Java case study where we analyzed the GitHub project with [urlhttps://github.com/json-iterator/java](https://github.com/json-iterator/java):

```
[
  {
    "file_path": "testRepo/java\\android-
demo\\src\\androidTest\\java\\com\\example\\myapplication\\ExampleInstrumentedTest.ja
va",
    "vulnerability_score": 0.9954152107,
    "vulnerability_flag": 1
  },
  {
    "file_path": "testRepo/java\\android-
demo\\src\\main\\java\\com\\example\\myapplication\\DemoCodegenConfig.java",
    "vulnerability_score": 0.9974604845,
    "vulnerability_flag": 1
  },
  {
    "file_path": "testRepo/java\\src\\main\\java\\com\\jsoniter\\Codegen.java",
    "vulnerability_score": 0.0,
    "vulnerability_flag": 0
  },
  {
    "file_path": "testRepo/java\\src\\main\\java\\com\\jsoniter\\CodegenAccess.java",
    "vulnerability_score": 0.0009024058,
    "vulnerability_flag": 0
  },
  {
    "file_path": "testRepo/java\\android-
demo\\src\\main\\java\\jsoniter_codegen\\cfg1173796797\\decoder\\int_array.java",
    "vulnerability_score": 0.9956971407,
```

```
"vulnerability_flag": 1  
}  
]
```

Figure 73: Model output (JSON format) for the Java case study

The useful information extracted by this model is that a vulnerability flag and a vulnerability score is assigned to each class file in the project. The vulnerability flag indicates the existence of vulnerabilities in the file and it can be equal either with zero (i.e. non vulnerable) or with one (i.e. vulnerable). The vulnerability score is the likelihood with which the model assigns these flags to the files. In other words, the score declares how sure the model is about its predictions. For example, in Cpp case study, the class file named Infinite Grid is not likely to contain vulnerabilities as it has flag equal to zero with score almost zero, whereas the class file named Gray Code needs further examination by its developers as it is considered vulnerable with 84% probability. So the users (i.e. developers) can utilize this model in order to prioritize their efforts on fixing vulnerabilities in their source code.

11.3.7 Conclusion and Future Work

In this study NLP techniques were used in the field of vulnerability assessment. Emphasis was given to the usage of BERT, a pre trained model that can be fine-tuned for a specific task. BERT models capable of predicting vulnerabilities in software classes were built. The models were built and evaluated on popular vulnerability datasets (in fact, benchmarks) curated by NIST and OWASP. By the evaluation of these models, we received an f2 score higher than 70%. The results of the evaluation indicate that BERT is a promising solution to be used as the basis of constructing vulnerability prediction models that deserves further investigation.

Several directions for future work can be identified. Future work includes the adoption of codeBert [50] which captures the semantic relationship between natural language and programming language and generates general-purpose representations that can be used to support natural/programming language understanding (e.g., natural language code search) and generation tasks. Furthermore, we could replicate our study to new datasets containing different programming languages such as Python and Javascript, in order to increase the generalizability of the approach. Last but not least, the adoption of software metrics that are statically extracted could be proved as a very beneficial process. We plan to examine if the voting of both text mining features and software metrics could be beneficial to the model's performance.

11.3.8 Security-related Static Analysis (SSA)

In most cases, software security is seen as an afterthought in the software development process. It is often introduced after software products have been implemented, mostly via the use of procedures designed to prevent malevolent persons from exploiting existing flaws (e.g., intrusion detection systems). However, the growing number of security problems disclosed each year suggests that these methods are unable to adequately secure software products from assaults. To that aim, software companies have changed their attention to developing software solutions that are extremely safe (i.e., as devoid of vulnerabilities as feasible) from the ground up.

A software vulnerability is a flaw in the design, development, or configuration of software that allows its exploitation to breach a security policy [83]. The majority of software flaws are caused by a small number of frequent programming mistakes [33]. These mistakes are created during the coding process by the developers, primarily owing to their lack of security knowledge [100] or to the fast production cycles [24]. However, expecting them to recall hundreds of security-related issue patterns and poor practices that they should avoid is impractical. As a result, effective tools are necessary to assist them in avoiding the introduction of such security flaws and, as a result, writing more secure code [157] [62].

Static analysis has been found to be useful in detecting security flaws early in the software development process [33]. Their major distinguishing feature is that they are applied directly to the system's source or generated code, without needing its execution [100]. In fact, static analysis has been found to be effective in uncovering security-related bugs early enough in the software development process [104] [106]. It is considered an important technique for adding security during the overall software development process. This belief is expressed by several experts in the field of software security (e.g. [104] [120]), while well-established secure software development lifecycles (SDLCs), including the well-known Microsoft's SDL [67] [94], OWASP's OpenSAAM, and Cigital's Touchpoints [135], propose the adoption of static analysis as the main mechanism for adding security during the coding (i.e., implementation phase) of the SDLC. In addition, ASA is a security activity commonly adopted by major technological firms including Google, Microsoft, Adobe and Intel, as reported by the BSIMM6 initiative. Hence, static analysis should be part of the software development process of security-critical software systems, like web applications, IoT software, etc.

Apart from their ability to detect potential security issues (i.e., vulnerabilities) that reside in the source code of a software product, static analysis tools can be used as the basis for getting more sophisticated and abstract information with respect to the broader security of the analyzed software. In fact, the outputs of static code analyzers, since they consist of extensive lists of raw warnings (i.e. alerts), they do not give significant insight to software product stakeholders (e.g., developers, engineers, project managers, etc.). Hence, proper knowledge extraction tools should be used that will be able to extract security-related information from these raw results and present them to the user in a more intuitive and easily understandable way (e.g., through aggregation, visualization, etc.). To this end, lately, the results of static analysis tools have been started to be used as the basis for the construction of models and techniques, able to aggregate the raw warnings produced by the tools in order to provide quantitative expressions of important security aspects of software applications [142]. Both the low-level warnings produced by static code analyzers, and their high-level metrics derived through their sophisticated aggregation are expected to facilitate the production of more secure software.

To this end, within the context of the SmartCLIDE project, we attempt to provide solutions for monitoring and improving the security of software applications (task- or workflow-level) based on static analysis properly configured to detect security issues. In particular, we performed a survey in the literature and identified the most suitable static code analyzers that can be used for detecting security issues. Then, we built a static analysis framework/platform, the Security-related Static Analysis Subcomponent (SSAS), which integrates the existing popular static code analyzers, which are properly configured to detect security issues. The framework (i.e., subcomponent) is highly customizable, allowing the user to select which types of security issues (e.g., Buffer Overflow, Cross-site Scripting, Denial of Service, etc.) it should search for while analyzing a desired software. The framework is also highly extendable, allowing the integration of additional static code analyzers in order to cover additional needs,

domains, and programming languages. Apart from the static analysis itself, in order to provide more high-level insights (e.g., at workflow level) and more abstract information with respect to the security level of the analyzed software, an aggregator module is also employed for the sophisticated aggregation of the static analysis results. For the construction of the aggregator module, state-of-the-art models and aggregation techniques have been utilized and extended in order to fit in the context of the SmartCLIDE project. The quantitative aggregated expressions are expected to provide useful insights to non-technical stakeholders, such as project managers, and therefore facilitate decision making during the overall software development process. The SSAS subcomponent is accessible through a REST API. It is demonstrated through case studies two on real-world open-source applications.

The rest of the section is structured as follows: Section 11.3.9 provides an overview of the related work in the field of static analysis for software security verification and validation. In Section 11.3.10 we present the proposed methodology that will be the basis of the Security-related Static Analysis Subcomponent. In Section 11.3.12 the proposed solution is demonstrated through two case studies on real-world open-source projects. Finally, Section 11.3.13 concludes the study and discusses directions for future work within the context of the SmartCLIDE project.

11.3.9 Related Work

Vulnerabilities are special types of software bugs with security implications [106]. They lead to the violation of one or more security policies, normally without influencing the functional requirements of the software products, constituting in that way their detection by conventional functional testing mechanisms highly difficult (if not impossible). According to [135], two types of software vulnerabilities exist: (i) design-level vulnerabilities, which may occur as a result of a design flaw, and (ii) implementation-level vulnerabilities, which may occur as a result of a bug in the code. A design-level vulnerability often manifest itself as a flaw in the code of the system (i.e., implementation-level vulnerability), if it is not eliminated during the early stages of the SDLC [120]. Therefore, as already mentioned, most of the software vulnerabilities stem from a small number of common programming errors [33], [67]. On average, roughly half of the software vulnerabilities are introduced during the coding phase [99]. It should be noted that well-known security breaches like the Equifax Breach [94] and the Heartbleed [29] were caused by simple implementation errors.

The majority of the issues that can be found by static analysis are common programming errors [172]. Since software vulnerabilities are normally caused by common programming errors, static analysis is considered a highly suitable and effective technique for detecting security vulnerabilities that reside in the source code of software products. In fact, ASA tools have been found effective in identifying security-related bugs early enough in the SDLC [33], [104], when their correction is relatively cheap and easy. For instance, the most representative example of vulnerabilities that can be detected by ASA tools is injection vulnerabilities (e.g. SQL Injection (SQLI) [63] and Cross-site Scripting (XSS) [55]), which are considered as the most common and dangerous threats of web applications according to both OWASP Top 10 [114] and CWE Top 25 [40] lists of most common software vulnerabilities. These vulnerabilities are commonly caused by unsanitized data flows from untrusted inputs to security sensitive sinks [49], which is a relatively simple but all too often overlooked code-level error. A multitude of ASA tools have been proposed for detecting injection vulnerabilities (e.g. [66], [10]), indicating the importance of static analysis in detecting such types of security issues. Finally, automated static analysis has been widely used in the literature for security purposes, including vulnerability detection (e.g., [74],

[101]), prediction (e.g., [105], [159], [76]), and automatic elimination (e.g., [101]) with promising results. According to a recent systematic mapping study in the field of Security by Design [104] , vulnerability detection, prediction and elimination using static analysis is the current trend in the field.

The results of the static analysis tools contain important information concerning the security of the analysed software. However, in their raw form, this information is hidden, and it is hard to be acquired and comprehended, especially by stakeholders with little (to no) technical expertise, like project managers. Appropriate knowledge extraction tools are needed, in order to extract the security-related information from the raw results produced by the static code analyzers, and present them to the users in a more intuitive and easily understandable form. One way of achieving this is through the conduction of security models, able to aggregate the raw results of the static code analyzers, in order to provide higher-level metrics that reflect security aspects of the analyzed software. In other words, the purpose of these models is to provide quantitative expressions of software security. These high-level scores are more intuitive and easily understandable, and can be used to facilitate decision making during the development of software products.

Several techniques and models for evaluating software security based on the results of static analysis have been proposed in the literature over the years [87], [4], [102], [37], [165], [45], [142]. However, none of them managed to provide highly reliable and operational solution to be used in practice. More specifically, highly automated models are unreliable because they are based on arbitrary parameters defined by their authors and not by international standards and security experts [87], [4], [102], whereas highly reliable models (i.e., models that are based on reliable source of information like security standards) are manual and impractical [37], [165], [45]. In addition to this, the vast majority of the existing solutions are not operational and cannot be used in practice. Recently, a security assessment model that is fully automated, operational, and its parameters were derived based on data and well-accepted sources of information (i.e., standards and security knowledgebases) has been proposed [142] . This model can be used as the basis for further extensions.

11.3.10 Methodology

The suggested methodology for the verification and validation of software security based on static analysis, is extensively detailed in this section. More specifically, the tools used to build the security analysis framework will be presented firstly. Following that, the general architecture of the overall Security-related Static Analysis Subcomponent (SSAS) will be described in detail. Finally, two examples will be presented on how to use our service.

11.3.11 Security Analysis Framework/Platform

The core element of our approach is a framework for conducting security-related static analysis on a given software application. This framework enables the execution of multiple static code analyzers for a given software product, and the reporting of potential vulnerabilities that may reside in its source code. The combination of several tools is highly necessary for enhancing the probability of detecting more security vulnerabilities. Different tools are able to detect different types of security issues and to support different programming languages. Hence, the main process followed by our framework is summarized as follows. For each programming language, the source code is analyzed and, depending on each tool and security standards it examines, detailed reports are produced that contain errors and possible security

issues in the code. Using such tools, the developer can immediately identify security vulnerabilities in the software implementation, have a detailed analysis of his code and the errors that occur as well, in combination with our model, an "indicator" of how secure his product is.

Hence, one important step of our work was to perform a survey of existing static code analyzers that could be utilized for verifying and validating the security level of software products. There are many static analysis tools that either cover a specific language or cover a variety of languages. Research has been conducted on the appropriate tools to integrate into the model covering a wide range of security issues as well as being extensible and applicable in the future to other programming languages. For the purposes of SmartCLIDE, initially the focus is directed to analyzing software products written in Java and Python programming languages. The tools that will be selected to be integrated into the framework should be able to detect problems in the source code, provide a detailed analysis of the code and the error, they should be able to suggest an indicative solution to the problem, they should be able to detect a significant range of security issues. The tools examined are presented in Table 17.

Table 17: Open-source and Commercial Static Code Analyzers

Tool	Description
CK7	CK calculates class-level and method-level code metrics in Java projects by means of static analysis (i.e. no need for compiled code). Currently, it contains a large set of metrics, including the famous Chidamber and Kemerer .
PMD	PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, Salesforce.com Apex and Visualforce, PLSQL, Apache Velocity, XML, XSL. Additionally it includes CPD, the copy-paste-detector. CPD finds duplicated code in Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective C, Matlab, Python, Go, Swift and Salesforce.com Apex and Visualforce.
SonarQube9	SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on 20+ programming languages. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities. SonarQube can record metrics history and provides evolution graphs. SonarQube provides fully automated analysis and integration with Maven, Ant, Gradle, MSBuild and continuous integration tools.
Pylint10	Pylint is a tool that checks for errors in Python code, tries to enforce a coding standard and looks for code smells. It can also look for certain type errors, it can recommend suggestions about how particular blocks can be refactored and can offer you details about the code's complexity.
Bandit11	Bandit is a tool designed to find common security issues in Python code. To do this Bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report. Bandit was originally developed within the OpenStack Security Project and later rehomed to PyCQA.

Tool	Description
Fortify12	Fortify SCA is a static application security testing (SAST) offering used by development groups and security professionals to analyze the source code for security vulnerabilities. It reviews code and helps developers identify, prioritize, and resolve issues with less effort and in less time.
Coverity13	Coverity is a fast, accurate, and highly scalable static analysis (SAST) solution that helps development and security teams address security and quality defects early in the software development life cycle (SDLC), track and manage risks across the application portfolio, and ensure compliance with security and coding standards.
Understand	Understand is a customisable integrated development environment that enables static code analysis through an array of visuals, documentation, and metric tools. It was built to help software developers comprehend, maintain, and document their source code.
Vera Code14	Veracode is an application security company based in Burlington, Massachusetts. Founded in 2006, the company provides a SaaS application security solution that integrates application analysis into development pipelines.

We decided to use open-source tools in order to be in line with the open source policy of the project. In addition to this, this would allow the community to have free access to our solutions enhancing in that way their practicality. The aforementioned tools were specifically chosen, as each one covers a separate field and by combining them will lead to better overall coverage on security issues. CK provides a variety of metrics for Java code, metrics that directly apply to code characteristics and can be included to security models. Moreover, CK produces detailed csv files that contain all the issues found by the tool and their location (file/ class, line of code, directory). We decided to allow the user to compute software metrics through our security analysis framework, since there are numerous findings in the literature that showcase the close relationship between software metrics and software vulnerabilities [141], [35], [143].

PMD is a static code analyzer that reports violations to specific rules that correspond to best practices. PMD lays out a set of rules, from which someone can build proper rulesets addressing security related issues. PMD includes rules for exception handling, misused functionality, null pointer, etc., rules that are grouped in order to form a ruleset for specific issues (Null Pointer ruleset, Exception Handling ruleset etc.). Moreover, for each specific rulesets, PMD provides a detailed csv file (NullPointer.csv, ExceptionHandling.csv) addressing the issues and the rules of each ruleset inside the source code. The PMD has already been used in the literature as the basis of security models (e.g., [142]).

SonarQube is a whole platform supporting 27 programming languages for the time being. It includes a variety of security profiles by default, giving also the opportunity to create profiles addressing specific security issues that the user will choose to include to the security profile. Providing and API, after completing the analysis SonarQube exposes proper endpoint to retrieve the analysis scores in JSON form as well as, by visiting the SonarQube site, a detailed analysis of the project analyzed with all the metrics and issues raised is presented in Sonar GUI, giving the user the ability to examine everything in detail. Pylint as well as Bandit are two static analyzers specifically for Python. Both empower Sonarqube coverage on security issues and complete the overall security coverage for Python projects.

At this point, we decided to integrate into the security analysis framework the CK, PMD, and SonarQube tools. Combining different tools, as already mentioned, provides better coverage of a larger spectrum of security issues that may reside in the analyzed software (e.g., Buffer Overflow, Denial of Service, etc.), increasing in that way the probability of detecting actual vulnerabilities. In fact, different metrics, rulesets, and issue categories (i.e., vulnerability types) are provided from each separate tool. As will be discussed later in this section, emphasis has been given on the configurability of the proposed framework. The user is equipped with the ability to cherry pick those metrics, rules, etc., from the supported tools that they find most interesting, important, or meaningful. This is important since different applications exhibit different characteristics and suffer from different types of security issues. In addition to this, it should be noted that the framework is easily extendable. This means that additional static code analyzers can be easily integrated into the existing framework in order to further expand the capabilities of the proposed security analysis platform. In the rest of the SmartCLIDE project, we are planning to also add tools for supporting the analysis of other programming languages including C/C++ and JavaScript.

11.3.12 Description of the Software Security Analysis Platform (SSAS)

After the selection of the most suitable static code analyzers, the next step is to build the service that corresponds to our security analysis framework/platform. The framework’s functionalities are exposed via a RESTfull API. In detail, the user is able to access the service via HTTP request. This request should contain the url of the project to be analyzed (e.g., GitHub, GitLab, Bitbucket, etc.), the language of the project to be analyzed (currently Python or Java), and a JSON containing a configuration of the selected tools, as well as of the aggregator (it will be explained later in the text). The overall architecture is suitable for scalability purposes. Specifically, our service is built on platform logic, where the user will be able to access it via HTTP requests, it will be easy to be included in a variety of projects as it’s functionalities are exposed via a RESTfull API and will be easy to extend and support more programming languages and tools included, since it is built in MVC logic, which means that for each tool there is a different sub-service for the code analysis and the extraction of the results. A high-level overview of the proposed security analysis framework (in fact, the SSAS) is provided in Figure 74.

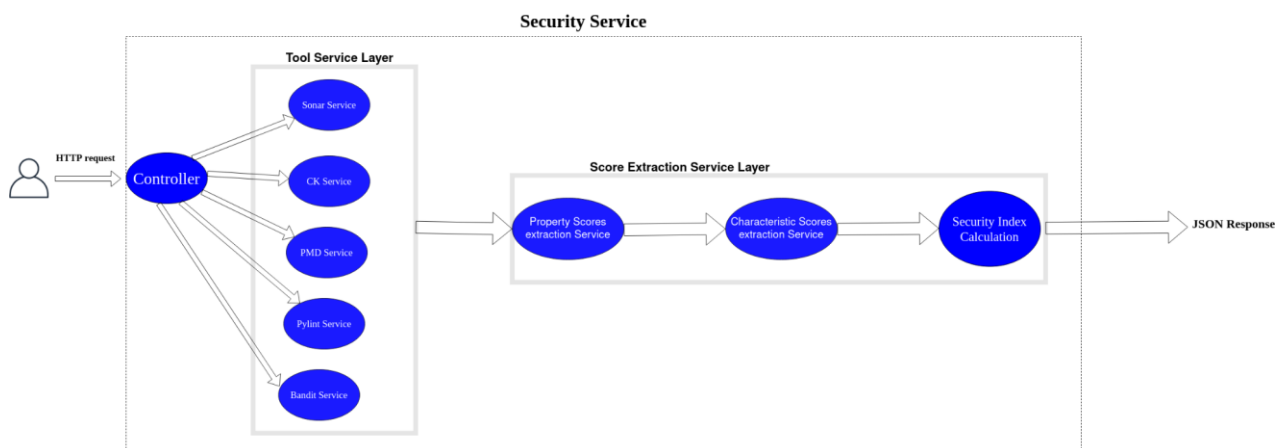


Figure 74: High-level overview of the security analysis framework/platform – The Security-related Static Analysis Subcomponent (SSAS)

As can be seen by Figure 74, an HTTP request needs to be applied to the SSAS. This request is parsed by a controller that clones the selected software project (i.e., software repository) locally, and configures the selected tools in order to be used for the analysis. After retrieving locally the repository of the selected

project, the selected static code analyzers are executed based on the programming language of the project and the configuration that was submitted by the user during the request. The service is built in MVC logic, meaning that a central controller handles all the requests of the service, and proper services are called in the backend to achieve the overall result. For each tool a different sub-service is built, which serves readability, maintainability and scalability for future expansion and support of more tools and languages. In brief, the controller invokes the selected static tools properly configured based on the user options in order to analyze statically the selected software project/repository and derive the results of the analysis.

After executing the static analysis tools, the raw results of the analysis are produced. As can be seen by Figure 74, the next step is the execution of the Aggregator module. The Aggregator module is responsible for aggregating the raw results produced by the static code analysis in a sophisticated way in order to produce higher-level metrics that reflect important security aspects of the analyzed software. For this purpose, the Aggregator module is based on state-of-the-art models that have proposed in the literature. More specifically, we utilized the concept proposed in [142], and we extended it in order to support additional programming languages, tools, and security issues (explained later in the text). According to the adopted aggregation approach, the first step corresponds to the evaluation of the selected security properties, i.e., low-level security aspects that reflect how free the code is from important security issues (e.g., Null Pointer, Denial of Service, etc.), which is reflected through the assignment of scores to these properties that lie in the $[0,1]$ interval. Then these scores are aggregated in order to derive the scores of high-level security characteristics/requirements, such as Confidentiality, Integrity, and Availability. The final security score of the overall project is derived by taking the weighted average of the scores of the security requirements. For more information about the overall aggregation approach, we refer the reader to [142].

When completing the analysis, the service returns a JSON containing the values of each property selected by each separate tool, the property scores, the characteristic scores and the overall security index. Moreover, by requesting in a different endpoint, the user has the ability to store locally a detailed csv report from all the tools used, containing issues on the source code, their exact location and in cases suggesting ways to fix those issues.

As mentioned previously, the security analysis platform is highly configurable. Apart from the selection of the specific tools and security issues that the framework should search for, the user can configure the aggregator through the submitted request. More specifically, the user can select the properties and the characteristics of the security model, along with the thresholds that are used for evaluating the properties, and the weights that are used for calculating the scores of the characteristics and the overall security score of the system. This is important as it allows the user to define their own models for analyzing the security level of a software, since it is known in the literature that no single model is enough for evaluating every kind of software. However, it should be noted that predefined security models have been formed based on our security expertise, as a default security profile to be used by the users for analyzing their software projects or to be utilized as a guide on how to build their custom models.

In the following, we present the main characteristics (i.e., security requirements) and properties (i.e., metrics and security issue categories that can be produced by the selected tools), in order to demonstrate to the user the full potentials of the proposed security framework. As far as the properties are concerned, we acknowledge two types of properties, namely alerts-based properties and metrics-based properties, following the same logic that is followed in [64] [142]. The metric-based properties are properties that

can be quantified through software metrics. The alert-based properties are properties that are quantified through the density of security-related static analysis alerts of the same category. The metric-based and alert-based properties that are supported by the security analysis framework are presented in Table 18 and Table 19 respectively.

Table 18: Metric-based Security Properties supported by the SSAS

Property	Description	Metric
Size	The size (i.e., volume) of the source code, expressed in number of instructions, number of methods, etc.	LOC (Lines of Code) Number of fields Number of Methods Number of visible methods
Complexity	The degree to which the software product's logic is complicated.	WMC (Weighted Methods per Class) DIT (Depth of Inheritance Tree)
Cohesion	The extent to which a software program adheres to the idea of separation of concerns.	LCOM (Lack of Cohesion in Methods) TCC (Tight Class Cohesion) LCC (Loose Class Cohesion) MNB (Max Nested Blocks)
Coupling	The degree of independence between the software product's parts.	CBO (Coupling Between Objects)

Table 19: Alerts-based Security Properties supported by the SSAS

Property	Description	Tool
Null Pointer	Contains rules for detecting issues with null pointer dereference. A NULL Pointer dereference usually results in a system crash.	PMD
Assignment	Contains rules for detecting variable assignment and declaration problems that have security consequences (e.g. local variables that are not declared final can be used as entry points by the attackers).	PMD
Exception Handling	Contains criteria for detecting inappropriate exception handling. Improper exception handling can cause a system crash or expose system information to users.	PMD
Resource Handling	Contains rules for detecting inappropriate management of system resources (e.g., memory, connections, and so on). Improper resource management might result in service deterioration or even service denial.	PMD
Logging	Contains rules for detecting improper use of logging capability. Incorrect logging may result in the omission of critical occurrences, whereas misplaced logging commands (e.g., within loops) may cause the program to run late.	PMD

Property	Description	Tool
Adjustability	Contains criteria for determining the presence of hard-coded security sensitive information. If the code is ever revealed, these hard-coded values may become available to attackers.	PMD
Misused Functionality	Contains criteria for detecting abused functions supplied by the programming language or commonly known APIs.	PMD
Cyclomatic Complexity	It is the Cyclomatic Complexity calculated based on the number of paths through the code. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each function has a minimum complexity of 1. This calculation varies slightly by language because keywords and functionalities do.	SonarQube
Code Smells	Code smells correspond to quality issues that affect the maintainability of a software product. The more code smells left unaddressed the less maintainable the code will be. Resent works have shown that this may have potential impact to security as well [143].	SonarQube
Bugs	Bugs correspond to code-level issues that are introduced during the programming phase of the overall software development lifecycle and can lead mainly to a software failure. A relationship between bugs and vulnerabilities has been observed in the literature [106], [143], whereas they obviously affect the Availability of a software system, which is an important Security Requirement according to ISO 27001 [69].	SonarQube
SQL-Injection	SQL Injection issues are one of the most popular types of security vulnerabilities. They allow an attacker to inject SQL commands allowing them to access and omodify sensitive data stored in SQL databases, affecting in that way the Confidentiality and Integrity security requirements of a software system.	SonarQube
Weak-Cryptography	Cryptographic hash algorithms such as MD2, MD4, MD5, MD6, HAVAL-128, HMAC-MD5, DSA (which uses SHA-1), RIPEMD, RIPEMD-128, RIPEMD-160, HMACRIPEMD160 and SHA-1 are no longer considered secure, because it is possible to have collisions (little computational effort is enough to find two or more different inputs that produce the same hash). In addition to this, improper configuration of strong cryptographic algorithms like AES can also lead to disclosure of sensitive information, infringing in that way the Confidentiality security requirement of a given software.	SonarQube
Denial of Service	Denial of Service vulnerabilities lead to the unavailability of a software system, not allowing legitimate users to use its features. An effective DoS attack infringes the Availability Security Requirement of the compromised software.	SonarQube
Insecure Configuration	Many of the vulnerabilities that a software product contains are caused by wrong configuration of specific frameworks and libraries that have to do with system security and communication.	SonarQube

Property	Description	Tool
Authentication Issues	Missing or ineffective authentication (and authorization) may allow malicious individual to access and modify sensitive information of legitimate users. This type of issues obviously affect the security requirements of Confidentiality and Integrity.	SonarQube

As already mentioned, the security analysis platform computes the properties presented in Table 18 and Table 19, based on the user options. In fact, the user, through the submitted request selects, which of the above mentioned properties should be calculated during the static analysis. Then, the properties are evaluated based on a set of thresholds in order to assign a security score, similarly to [142] and [151]. The user can also define the Security Characteristics (i.e., the Security Requirements) that the Aggregator should focus on. The user can define any Security Characteristic they wish. We highly recommend the users to choose security characteristics from Table 20, since they are proposed by two international standards, namely ISO/IEC 25010 [70] and ISO/IEC 270001 [69].

Table 20: Recommended Security Characteristics (i.e., Requirements) to be used in the analysis performed by the SSAS

Characteristic	Description
Confidentiality	The extent to which the software product ensures that data is only available to those who have been granted access.
Integrity	The extent to which a software product, system, or component prevents illegal data alteration.
Availability	The extent to which a system, product, or component is operational and usable when needed.
Non-repudiation	The extent to which a user cannot repudiate actions that they have performed.
Authentication	The degree to which the identity of a subject or resource can be proved to be the one claimed.
Authorization	The degree to which appropriate access control has been granted to specific entities through assignment of required privileges.
Security compliance	The degree to which the software product adheres to standards, conventions or regulations relating to security.

In this section, we showcased the properties (i.e., code-level security aspects that a software application may exhibit) and the characteristics (i.e., security requirements that a software application should satisfy) that are currently supported by the proposed security analysis framework/platform (in fact, the SSAS). As already mentioned, the framework is highly configurable allowing the user to select the tools that should be executed and the security issues (i.e., properties) that the analysis should focus on. It also allows the user to define the parameters of the Aggregator, which are the security characteristics (i.e., security requirements of interest), as well as the parameters that are required for the aggregation (these are described in the next section). Default models, based on our expertise have been produced and can be

used as the default option for security analysis. However, the high configurability of the security analysis framework/platform (SSAS) enables the user to modify these models in order to better fit their specific needs, or even to create custom models (i.e., define their own preferable configuration to the static code analyzers and Aggregator).

11.3.13 Security Model and Parameter Selection

The second feature (step) of the security analysis framework, as can be seen by Figure 74, is the Aggregator, which is responsible for aggregating the low-level results produced by static analysis, into higher-level metrics (i.e., security scores). Two are the main parameters of the Aggregator, namely the thresholds, which are used for assigning scores to the selected properties, and the weights, which are used to compute the scores of the characteristics and the overall security score of the system. These parameters are configurable, meaning that the user can define them upon the HTTP request. To facilitate the usability of the security analysis framework/platform (SSAS) and to avoid the time-consuming step of constructing custom models, and mainly, of determining the model parameters, we provide two security models, one for Java and one for Python software applications. In the following we present the main structure and parameters of these two models.

For the model that is meant for analyzing Java applications, we selected 15 properties, which are Exception Handling, Assignment, Logging, Null Pointer, Resource Handling, Misused Functionality, Complexity, Coupling, Cohesion, SQL Injection, Vulnerabilities, Denial of Service, Insecure Configuration, Authentication Issues, and Weak Cryptography. For the model that is meant for analysing Python applications, we selected 7 properties, which are Complexity, Denial of Service, Vulnerabilities, Weak Cryptography, Insecure Configuration, Authentication Issues. The selection was based on the existence of corresponding rulesets for these programming languages at the selected static code analyzer, as well as on the ability to compute a reliable set of thresholds (explained later in this section).

For the selection of the thresholds of the models, the benchmarking approach, a popular approach for determining thresholds for software metrics, has been utilized. In particular, we followed an approach similar to [142] and [64]. According to this approach, a benchmark of representative software applications is constructed and analyzed in order to compute the desired metrics. Then the thresholds are computed based on the statistical distribution of the computed metrics. More specifically, the minimum, median, and maximum values of each metric are chosen as the lower (tl), middle (tm), and upper (tu) thresholds respectively.

To this end, we built a crawler able to retrieve software repositories from GitHub. More specifically, we used the crawler to download the 100 most popular open-source software applications for Java and the 100 most popular software applications for Python. As a measure of popularity we used the number of GitHub stars that the repositories have. Then we applied the aforementioned approach for computing the thresholds of their properties. The selected thresholds of the Java and Python models are illustrated in Table 21 and Table 22 respectively.

Table 21: The thresholds of the properties of the security model for analyzing Java projects

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
tl	0	0.01	0.13	0	0	0	0	0	0	0	0	0	0	0	0
tm	0.10	0.03	0.04	0.22	0.11	0.05	0.32	2.20	0.13	0.09	0.01	0.02	0.00	0.02	0.73
th	3.184	0.571	0.276	12.98	7.692	6.849	25.96	166.6	4.784	4	1.547	2.217	0.198	3.095	32.05

P1 Cohesion, P2 Coupling, P3 Complexity, P4 Exception Handling, P5 Assignment, P6 Logging, P7 Null Pointer, P8 Resource Handling, P9 Missused Functionality, P10 Vulnerability P11 SQL Injection, P12 Denial of Service, P13 Weak Cryptography P14 Authentication Issues, P15 Insecure Configuration

Table 22: The thresholds of the properties of the security model for analyzing Python projects

	P1	P2	P3	P4	P5	P6	P7
tl	0.022	0	0	0	0	0	0
tm	0.027	0.098	0.013	0.244	0.001	0.024	0.735
th	0.308	4	1.547	2.217	0.198	3.095	32.051

P1 Complexity, P2 Vulnerabilities, P3 SQL Injection, P4 Denial of Service, P5 Weak Cryptography, P6 Authentication Issues, P7 Insecure Configuration

As far as the weights of the models are concerned we utilized the novel weights elicitation approach that was introduced in [142]. In brief, this approach is based on popular decision making techniques in order to produce a set of weights that reflect the knowledge that is expressed by the Common Weakness Enumeration (CWE)15knowledge base. The produced weights of the Java and Python models are illustrated in Table 23 and Table 24 respectively.

Table 23: The weights of the security model for analyzing Java projects

Property Name	Confidentiality	Integrity	Availability
Cohesion	0.005	0.01	0.005
Coupling	0.005	0.005	0.005
Complexity	0.005	0.005	0.01
Exception Handling	0.1	0.1	0.1
Assignment	0.1	0.15	0.01
Logging	0.1	0.01	0.01
Null Pointer	0.01	0.01	0.2
Resource Handling	0.01	0.01	0.3

Misused Functionality	0.01	0.01	0.01
Vulnerability	0.1	0.15	0.01
SQL Injection	0.1	0.15	0.01
Denial of Service	0.005	0.01	0.3
Weak Cryptography	0.2	0.16	0.01
Authentication Issues	0.15	0.21	0.01
Insecure Configuration	0.1	0.01	0.01

Table 24: The weights of the security model for analyzing Python projects

Property Name	Confidentiality	Integrity	Availability
Complexity	0.05	0.05	0.05
Vulnerabilities	0.2	0.2	0.15
SQL Injection	0.1	0.2	0.05
DoS	0.05	0.05	0.4
Weak Cryptography	0.3	0.1	0.05
Authentication Issues	0.2	0.3	0.1
Insecure Configuration	0.1	0.1	0.2

At this point it should be noted that the users can change the values of the thresholds and weights of the produced models. In addition to this, they can adopt any other threshold and weights elicitation techniques for their custom models.

11.3.14 Case Studies

In this section, the proposed security analysis framework/platform is demonstrated through two case studies on real-world open-source software products. More specifically, two examples are provided, one for Java and one for Python projects. The main goal of these case studies is to help the reader understand how the proposed solution works and what benefits are expected. The whole process is fully explained and demonstrated in order to provide an overall view of our approach.

As mentioned before, our model’s functionality is exposed via a REST API. Through POSTMAN¹⁶ we will demonstrate how to properly call our services, what is the type of the request our endpoint accepts, the response of the request as well as the functionality behind the calls. The structure of the request explained below and in Table 25:

Table 25: The parameters of the HTTP Request that should be submitted for analyzing a specific software project using SSAS

Parameter Name	Description
url	The url of the project’s repository to be analyzed.
properties	A JSON containing proper software properties alongside with their thresholds from the tools provided(besides sonarqube) as well as characteristics alongside with their weights defined. This is the initial version of the service and, in the future, characteristics will be given in a separate JSON.
sonarqube	A JSON containing the metric keys(as Sonarqube defines them) and the vulnerabilities, which are density based properties. Sonarqube separates security issues into two categories. We, then, need a separate object from the above to properly call the Sonarqube API. As in the previous one, this JSON also contains the thresholds for each metric key or vulnerability.
language	Indicating the implementation language of the project. For this version, it could be either Java or Python.

Below is a printscreen of the POSTMAN request for analyzing a Java project through the SSAS using the default model that we presented in Section 11.3.13. This is a guide for anyone who wants to understand the structure of the request and use our platform.

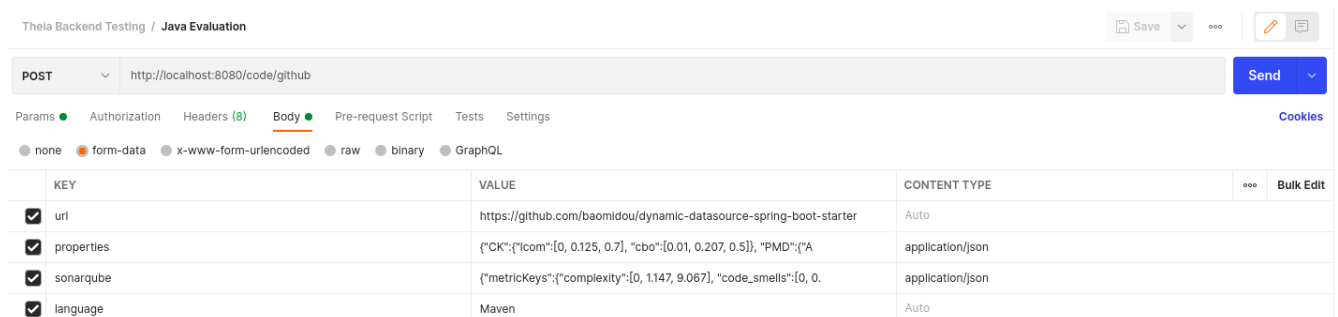


Figure 75: A screenshot of the HTTP Request submitted fro analyzing the security of an open-source Java project.

Properties and Sonarqube JSON’s are structured as shown in Table 26.

Table 26: The parameters of the HTTP Request that were used for analyzing the selected open-source Java project

Parameter Name
properties
<pre>{ "CK":{ "lcom":[0,0.10910936800871021,3.1849529780564265], "cbo":[0.017050298380221655,0.03692993475020107,0.5714285714285714], "wmc":[0.13793103448275862,0.04986595433654195,0.2765273311897106]}, "PMD":{"ExceptionHandling": [0,0.22938518010164352,12.987012987012987], "Assignment": [0,0.11160028050045478,7.6923076923076925], "Logging": [0,0.05692917472098835,6.8493150684931505], "NullPointer": [0,0.32358608981534065,25.966183574879228],</pre>

<pre>"ResourceHandling":[0,2.201831659093579,166.66666666666666],"MisusedFunctionality":[0,0.13732179935769162,4.784688995215311],"Characteristics":{"Confidentiality":[0.005,0.005,0.005,0.1,0.1,0.1,0.01,0.01,0.01,0.1,0.1,0.005,0.2,0.15,0.1],"Integrity":[0.01,0.005,0.005,0.1,0.15,0.01,0.01,0.01,0.01,0.15,0.15,0.01,0.16,0.21,0.01],"Availability":[0.005,0.005,0.01,0.1,0.01,0.01,0.2,0.3,0.01,0.01,0.01,0.3,0.01,0.01,0.01]}</pre>
<p>SonarQube</p> <pre>{"metricKeys":{"vulnerabilities":[0,0.09848484848484848,4]},{"vulnerabilities":{"sql-injection":[0,0.013234192551328933,1.5479876160990713],"dos":[0,0.024419175132769335,2.2172949002217295],"weak-cryptography":[0,0.0015070136414874827,0.1989258006763477],"auth":[0,0.024207864640426638,3.0959752321981426],"insecure-conf":[0,0.7356100591012389,32.05128205128205]}</pre>

As can be seen the Table 26, in these two parameters, namely properties and sonarqube, all the required configuration that the SSAS needs in order to analyze the selected software project based on the Java model. More specifically, for each tool, i.e., CK, PMD, and SonarQube, the low-level properties are selected, along with their thresholds. In addition to this, in the propertiesparameter the characteristics of the model are also defined along with the selected weights. These value are defined in the form of JSON Strings. Hence, this shows that through the submitted request, the user can define everything that is needed by the security analysis framework/platform (SSAS) for the static analysis and aggregation.

After the analysis is complete, the service returns a JSON file with the results of the analysis. The response that is produced by SSAS for the software repository that was used in the present example is illustrated in Table 27.

Table 27: The results of the analysis of the selected open-source Java project

Parameter Name	Description
response	<pre>{ "CK": { "loc": 2900.0, "cbo": 0.12758620689655173, "lcom": 0.06275862068965517, "wmc": 0.28 }, "PMD": { "Assignment": 0.3448275862068966, "Logging": 0.0, "NullPointer": 0.0, "MisusedFunctionality": 0.6896551724137931, "ResourceHandling": 0.6896551724137931, "ExceptionHandling": 1.3793103448275863 } }</pre>

	<pre> }, "Sonarqube": { "insecure-conf": 2.0811654526534857, "auth": 0.0, "ncloc": 3844.0, "weak-cryptography": 0.2601456815816857, "vulnerabilities": 4.0, "dos": 0.0, "sql-injection": 0.0 }, "Property Scores": { "Logging": 1.0, "NullPointerException": 1.0, "MisusedFunctionality": 0.4405756689993835, "insecure-conf": 0.47851626185942586, "auth": 1.0, "lcom": 0.7124049848559055, "weak-cryptography": 0.0, "ExceptionHandling": 0.4549318579390214, "dos": 1.0, "wmc": 0.0, "sql-injection": 1.0, "Assignment": 0.4846170487108379, "cbo": 0.41519503893430676, "ResourceHandling": 0.843390576757875, "vulnerabilities": 0.0 }, "Characteristic Scores": { "Availability": 0.39383196087637506, "Confidentiality": 0.557815001140017, </pre>
--	---

	<pre> "Integrity": 0.63511727079464 }, "Security index": { "Security Index": 0.5289214109370106 } } </pre>
--	--

As shown in Table 27, the first part of the JSON consists of the normalized values of the properties selected from each tool. Afterwards, the “Property Scores” includes the property scores as calculated by the utility function and the thresholds per each property, according to the equations described in [142] and [64]. Following that, “Characteristics Scores” included as calculated by the multiplication of the weight and the property score per each characteristic. Finally, the overall “Security Index” of the project analyzed is included, containing the security score as it is calculated by the characteristic scores.

In the given example, the Security Index of the analyzed software was found to be 0.53, which indicates that it is of average security and probably it needs improvements. By having a look at the Security Characteristics, we can see that the lowest score is assigned to Availability, which is 0.39. This indicates that the project may have issues that may affect the availability of the project, like exception handling issues. Hence, the high-level scores may facilitate decision making during implementation, by deciding which types of issues to fix first. It should be noted that the detailed results of the analysis (i.e., metrics and alerts) can be also retrieved from the security analysis framework/platform (SSAS) through the submission of the appropriate request.

The same approach is followed for the analysis of Python projects. The only parametrization that changes is the properties, including the proper tools for the python analysis as well as the url of the project and the language, which should be declared as Python. Below is a printscreen of the POSTMAN request. This is a guide for anyone who wants to understand the structure of the request and use our platform.

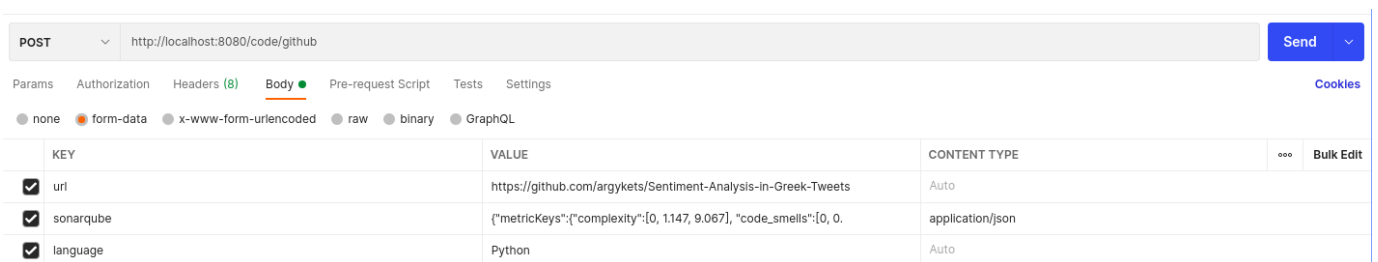


Figure 76: The HTTP request that should be submitted for analyzing the selected open-source Python Project

Similarly to the previous example, for the analysis of the Python project, we use the model that was derived in Section 11.3.13. The parameters that need to be provided for properly configuring the security analysis framework/platform (SSAS) to analyze the selected project using the desired model is presented in Table 28.

Table 28: The parameters of the request that were submitted for analyzing the selected open-source Python Project

Parameter Name
properties
<pre>{"Characteristics":{"Confidentiality":[0.05,0.2,0.1,0.05,0.3,0.2,0.1],"Integrity":[0.05,0.2,0.2,0.05,0.1,0.3,0.1],"Availability":[0.05,0.15,0.05,0.4,0.05,0.1,0.2]}}</pre>
SonarQube
<pre>{"metricKeys":{"complexity":[0.0227272727272728,0.027435805175993106,0.3082355281480008],"vulnerabilities":[0,0.0984848484848484,4],"vulnerabilities":{"sql-injection":[0,0.013234192551328933,1.5479876160990713],"dos":[0,0.024419175132769335,2.2172949002217295],"weak-cryptography":[0,0.0015070136414874827,0.1989258006763477],"auth":[0,0.024207864640426638,3.0959752321981426],"insecure-conf":[0,0.7356100591012389,32.05128205128205]}}</pre>

The response of the service, after code analysis, property scores extraction for both numerical and density-based properties, characteristic scores and calculation of the security index is illustrated in Table 29.

Table 29: The results of the analysis of the selected open-source Python project

Parameter Name	Description
response	<pre>{ "Sonarqube": { "complexity": 0.1942103697828886, "insecure-conf": 0.0, "auth": 0.1982750074353128, "ncloc": 10087.0, "weak-cryptography": 0.0991375037176564, "vulnerabilities": 0.0, "dos": 0.0, "sql-injection": 0.0 }, "Property Scores": { "complexity": 0.20303645095917502, "insecure-conf": 1.0, "auth": 0.47166661371670177, "weak-cryptography": 0.2527325247446452, </pre>

	<pre> "vulnerabilities": 1.0, "dos": 1.0, "sql-injection": 1.0 }, "Characteristic Scores": { "Availability": 0.6348281631316519, "Confidentiality": 0.8699551101568611, "Integrity": 0.8171217715285314 }, "Security index": { "Security Index": 0.7739683482723481 } } </pre>
--	--

For the time being the idle security model for the Python supports SonarQube properties. In the future open source tools such as Bandit and Pylint will be added to enhance security aspects for the Python language. Again, as shown in the response, the first part of the JSON consists of the normalized values of the properties selected from SonarQube. Afterwards, the “Property Scores” includes the property scores as calculated by the utility function and the thresholds per property. Following that, “Characteristics Scores” included as calculated by the multiplication of the weight and the property score per each characteristic. Finally, the overall “Security Index” of the project analyzed is included, containing the security score as it is calculated by the characteristic scores.

11.3.15 Conclusion and Future Work

To sum up, in the present section we presented our work towards the construction of a service-based security analysis framework/platform, the Security-related Static Analysis Subcomponent (SSAS) as defined in the SmartCLIDE architecture, which is able to analyze the security of software products based on static analysis. The platform currently supports analysis of Java and Python programming languages, with several open-source static code analyzers to be integrated. Our goal is to create a platform that is easily extensible with more open-source tools, and fully customizable, so the user could evaluate his projects based on the needs of his implementation. The platform also aggregates the raw results produced by the static analysis tools in order to compute higher-level security metrics using an Aggregator Component, which is based (and actually extends) novel concepts from the field of software security and quality evaluation [142]. We’ve selected a subset of tools and created some prototype security evaluation models to demonstrate our initial implementation of the platform, as well as to be used directly by stakeholders for assessing the security of software products, without having to pass through the time-consuming and effort-demanding step of model definition. In fact, those models can be used directly by

D.2.1 SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment

the developers of the SmartCLIDE to evaluate their services and correct any security flaws that may come up.

In the future, we are planning to further extend the security analysis platform by adding more open source tools and analyzers, in order to identify more types of security issues and to support additional programming languages like C, C++, and JavaScript. We are also planning to work towards visualizing effectively the assessment results produced by the SSAS, in order to provide better insights to the stakeholders. This will be achieved through our technical work with respect to workflow evaluation, and particularly towards the integration of our services into the broader SmartCLIDE Platform.

12 Services and Workflows Deployment & CI/CD

Modern development and operation (DevOps) processes are typically related with two main topics: Continuous Integration (CI), and Continuous Delivery (CD). Although both concepts are interrelated, there is still some misunderstanding about their goal, and their relations. Both CI/CD procedures optimise software delivery times, which is highly appreciated by developers and software companies. Thanks to modern CI/CD tools and by using well-known Version Control Systems (e.g., Git), code changes can be dynamically detected from repositories, enabling automated pipelines for building, testing, and even deploying to a chosen environment. Despite the similar nature and goal of CI/CD engines, provided solutions vary in the way they provide these capabilities and the required configuration (e.g., to define pipelines).

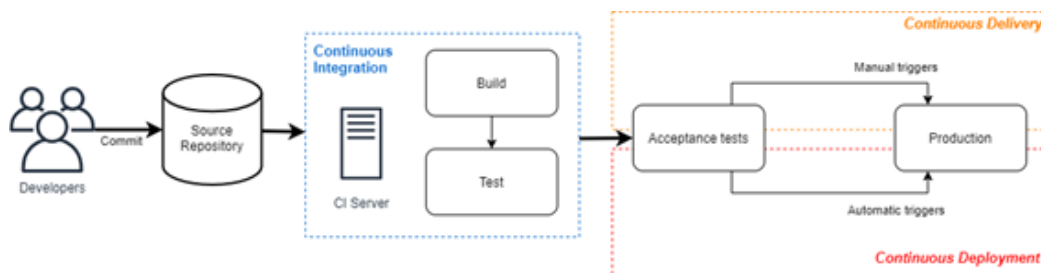


Figure 77: Continuous Integration, Delivery and Deployment

Continuous Integration and Deployment can be also applied to modern microservice-based applications, commonly related to containerization practices. In such a case, the CI/CD workflow must be compliant with building technologies (e.g., Docker), and also remote deployment techniques, such as clustering (e.g., Kubernetes) or third-party cloud providers (e.g., AWS). However, CI/CD configuration may get complex, especially by considering that different CI/CD engines could be used by the same developer so she has to carry differences in syntax (e.g., configuration files).

To mitigate this, SmartCLIDE will assume CI/CD tools as an agnostic element, providing an adaptation layer between the user configuration and a generic chosen CI/CD tool. This adaptation layer and its connection is named the Services Deployment (SD) component. The Services Deployment component may receive configuration files and docker images as an input, and will outcome a deployed service at the chosen remote machine (e.g., Kubernetes, AWS). This process should be compliant with any CI/CD tools in a generic way. In this document, we will study one of them, GitLab, able to handle not only CI/CD, but also providing Git and docker image repositories.

12.1 Innovative Approach to Enable Agnostic CI/CD tools

There are several CI/CD tools and integrated solutions within the technology market, some of them available under paid license and other ones available as open source tools/solutions. Almost all existing solutions are designed under concrete use cases or have very narrow feasible and useful scenarios, so to reach a real agnostic CI/CD tool, which could be useful to deploy a defined application in any infrastructure or cloud provider is quite difficult. In that sense, the most common way to solve this issue is to deploy a complex chain of tools which collect the proper info and files, prepare all configuration steps, create the docker/kubernetes/container files and deploy them into the selected, pre-configured by

developer environment. In many cases, this process will entail many errors or inconsistencies due to the number of intermediaries and transformations before the deployment. In other cases, the CI/CD solution is so intricate that a minor change means a complete redesign.

SmartCLIDE CI/CD component proposes a novel, fully integrated solution for continuous deployment for both deployment cases: start-from-zero deployments and cumulative improvements, CI/CD component will receive only the proper, minimum information for deployment from Service Creation and Composition component and will make all necessary transformations on-the-fly in order to deploy the application properly in a previously designated environment. An interpreter within the CI/CD component will prepare the configuration file for the destination interpreter (changing syntax or commands within the code lines). In addition, all dockers/kubernetes/containers will be created and piped down to the selected destination environment.

This approach makes the task of continuous delivery much more friendly and affordable to the developer, reducing quite noticeably the rework, redesign and piping definition efforts and defining a real agnostic CI/CD solution.

12.1.1 State of the art

In this section a comparison between main existing solutions will be presented, where pros and cons are shown in order to collect the most beneficial features of each one and perform a best-in-class CI/CD solution with the SmartCLIDE CI/CD component. Licensed solutions have been removed from the comparison due to the project’s main aim: an open source tool for developer empowerment and creation of a single-stop development solution for all stages. Table 30 provides a summary of the most important tools found in the literature.

Table 30: State of the Art of CI/CD tools

	Commit	Build	Deploy		Config, syntax	Free to use	Open source
	Git	Docker	Kubernetes	AWS			
GitLab CI	Native	Runner	Runner	Runner	Yaml	Yes	Yes
Jenkins	Plugin	Plugin	Plugin	Plugin	Jenkins Pipeline (custom Groovy)	Yes	Yes
GitHub Actions	Native	Action	Action	Action	Yaml	Yes	Yes
CircleCI	GitHub	Native	Orbs	Orbs	Yaml	Pay per use	No

GITHUB ACTIONS

Github actions is a compound of functionalities within Github that automates all software development workflows, intended to build, test and deploy solutions.

Pros

- Native integration with Github repository
- Docker support
- Ability to duplicate workflows
- Ready-to-use actions, available within Github marketplace
- Fast and low latency
- Backed up with a growth and improvement roadmap

Cons are like:

- It uses to work with its own format (.github files), which makes difficult to interact outside Github environment
- CI/CD is not performed by Github itself, but by third-parties applications. This could entail complex solution chains and increase the error margin
- Github and Github actions performs better and more robust when the deployment environment is an Azure one
- Owned configuration format GITLAB CI

GITLAB

GitLab offers a continuous integration service. If a .gitlab-ci.yml file is added to the root directory of the repository, and the GitLab project is configured to use a Runner, then each merge request or “push” triggers the defined Gitlab CI pipeline. Gitlab CI is integrated natively with 15 third-parties tools in addition to its own CI engine.

Pros

- Robust CI with docker support
- Source control and CI in one place
- Open source and free to use application
- Easy set up and dedicated runner
- Integrated VCS on commit
- Hosted internally or self-hosted
- All in one solution
- Built-in support of kubernetes
- Built-in docker registry
- Built-in support of Review Apps
- Pipelines could be started manually

Cons

- According to user reports, if delivery takes more than 20 minutes, Gitlab CI fails the deployment
- GUI is often confusing or too complex to understand
- Sometimes some features are not available due to internal maintenance or similar and there is no warning messages about that

JENKINS

In a nutshell Jenkins CI is the leading open-source continuous integration server. Built with Java, it provides over 300 plugins to support building and testing virtually any project thanks to its great scale of integration (over 140 third-parties applications connected).

Some major pros could be:

- Hosted internally and externally
- Open source and free to use licensing
- Configuration as code
- Great performance building, testing and deploying any application asynchronously
- Self-hosted Gitlab integration
- API available for AWS, docker and kubernetes

And, in the other hand, some main cons are:

- Depending on the task to be automated, Jenkins can result difficult to configure properly
- Error reporting is not as accurate as needed
- Jenkins is not cloud based nowadays
- The infrastructure needed for Jenkins execution must be maintained by user
- Strong dependency of plugins and third-parties developments

12.1.2 Vision of the Technical Approach

Taking into account the current stage of the Smartclide project, the most appropriate alternative of those described above is Gitlab, because of its features of self-hosted and built-in support for both Docker and Kubernetes. Gitlab will act both as a repository and as publisher to the CI/CD engine chosen by user-developers. In the middle, the brand new Smartclide CI/CD component will assess the configuration file syntax and will translate/adjust the syntax in order to make the code fully compatible with the destination CI/CD engine. After that, the docker files (containers in general) will be created by the selected CI/CD engine and sent back to the Gitlab repository, where they may be used for deployment in a cloud environment asynchronously. In this way, an agnostic continuous integration and continuous deployment is possible thanks to the Smartclide' syntax translator packed inside Services Deployment component, which makes compatible and reachable any CI/CD engine from a common source and, in a final stage, would deploy in several cloud environments using as many CI/CD engines as needed.

A brief schema is shown below for better understanding. Note that schema shows a final/definitive workflow, where Smartclide Service Deployment component has to reach.

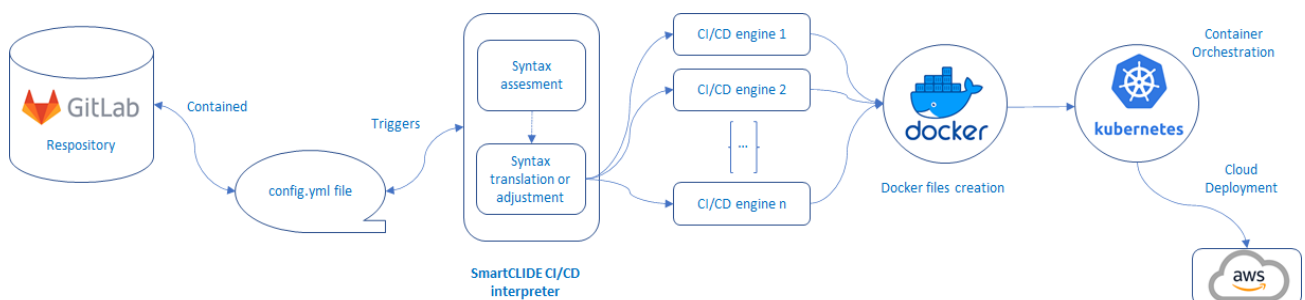


Figure 78: Brief schema of a Workflow

This is a very first approach where we will only consider:

GitLab + “SmartCLIDE Interpreter + Docker + Kubernetes/AWS

The need of the Services Deployment component, how this wrapper/adaptation will allow the developer to use its preferred CI/CD solution in SmartCLIDE. It will bridge the gap that exists between development and operations teams, greatly increasing efficiency in the software development and implementation process, and allowing the developer to take the worry out of the continuous delivery and deployment process. Most of CI/CD tools use a similar information/syntax at the configuration file .yaml. Our syntax will use YAML format, which acts as a descriptor for its implementation: Services Deployment will act as an intermediate by adapting these inputs to the chosen CI/CD engine.

12.1.3 Conclusion and Future Research

The sections collected an early description of the document. The research and development process in the next phases will allow us to deepen into the components of the deployment services as well as the characteristics that the interpreter of the deployment services and external transfer tools must meet.

As the prototype development progresses, we will update the content of this section, where we will collect the syntax structure and interoperability possibilities.

13 Conclusions

Deliverable D2.1 “*SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment*” of the SmartCLIDE project outlines the novel approaches and technologies to be used in the SmartCLIDE framework. In particular, the deliverable documents novel approaches for service discovery, creation, composition, and deployment and details on the adoption of existing technological approaches for service discovery, creation, composition, and deployment.

The SmartCLIDE platform will support service classification and discovery based on metadata/source code, leveraging the power of AI-based approaches. The platform will entail a service registry allowing for quick searches based on textual descriptions. Apart from the reuse of existing services the SmartCLIDE approach supports the creation of new services which can be composed (along with existing services) to model business processes adopting BPMN notation. The platform will offer to the users a template-based code generation approach which along with the SmartAssistant component supporting the selection and application of architectural and security Patterns will guide the end user during application development. Innovative approaches for extracting textual summaries of service workflows and for providing recommendations during service composition are also supported. The research approach that will be followed to support the testing of cloud applications developed through SmartCLIDE is presented in this deliverable. Furthermore, the approaches that will be employed for statically assessing the maintainability, reusability, and security of service-based software projects are thoroughly described. Finally, a key aspect of the envisioned SmartCLIDE platform is the automated and continuous deployment of services and workflows. The available technologies are analysed, explaining the rationale for choosing Gitlab as a repository and Kubernetes as the container orchestrator. All relevant technologies and the corresponding state-of-the-art are outlined in the sections of the current deliverable.

The final version of the aforementioned initial description of approaches will be outlined in deliverable D2.2 “Final SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment” which will contribute among others the service specification for runtime monitoring and verification, source code autocomplete suggestions, approaches for requirements assessment and approaches for service monitoring upon deployment.

References

- [1] R. Akkiraju, J. Farrell, J. A. Miller, M. Nagarajan, A. P. Sheth, and K. Verma, K., “Web service semantics-wsdl-s”, 2005.
- [2] M. Allamanis and C. Sutton. “Mining idioms from source code”. In *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*, pages 472–483, 2014.
- [3] G. Alonso, F. Casati, H. Kuno, V. Machiraju, “Concepts, Architectures and Applications”. *Web Services; Springer*: Berlin/Heidelberg, Germany, 354, 2004.
- [4] B. Alshammari, C. Fidge, and D. Corney, “A Hierarchical Security Assessment Model for Object-Oriented Programs,” *11th Int. Conf. Qual. Softw.*, no. 1, pp. 218–227, 2011.
- [5] N.S.R. Alves, T.S. Mendes, M.G. de Mendonça, R.O. Spínola, F. Shull, and C. Seaman, “Identification and management of technical debt: A systematic mapping study”, *Information and Software Technology*, 70, Elsevier, 2016, pp. 100–121.
- [6] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun, and K. Systa. 2016. “The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study”. *8th International Workshop on Managing Technical Debt (MTD '16)*. IEEE, Raleigh, NC, USA, 9-16
- [7] A. Ampatzoglou, S. Charalampidou, and I. Stamelos, “Research state of the art on GoF design patterns: A mapping study”, *Journal of Systems and Software*, 86 (7), pp. 1945-1964, 2013.
- [8] A. Ampatzoglou, S. Bibi, A. Chatzigeorgiou, P. Avgeriou, and I. Stamelos, “Reusability Index: A Measure for Assessing Software Assets Reusability”, *17th International Conference on Software Reuse (ICSR'18)*, Spain, 21-23 May 2018.
- [9] G. Artha A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo. “Categorizing the content of github readme files”. *Empirical Software Engineering*, 24(3):1296–1327, 2019.
- [10] S. Arzt et al., “FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps,” *35th ACM SIGPLAN Conf. Program. Lang. Des. Implement. - PLDI '14*, pp. 259–269, 2013.
- [11] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, 2016. “Managing technical debt in software engineering” (dagstuhl seminar 16162). In *Dagstuhl Reports* (Vol. 6, No. 4).
- [12] M. A. Babar, X. Wang, and I. Gorton, “Supporting Security Sensitive Architecture Design”. *QoSA-SOQUA 2005*. 2005: Springer-Verlag.
- [13] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” arXiv Prepr. arXiv1409.0473, 2014.
- [14] M. Balog, A. L Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. “Deepcoder: Learning to write programs”. arXiv preprint arXiv:1611.01989, 2016.
- [15] J. Bansiya and C. G. Davis, “A hierarchical model for object-oriented design quality assessment,” *IEEE Transaction of Software Engineering*, vol. 28, no. 1, pp. 4–17, Jan. 2002.
- [16] L. Bass, P. Clements, and R. Kazman, “Software Architecture in Practice”. 2nd ed. 2003, Addison-Wesley.
- [17] B. Bauer, J. P. Müller, J. Odell, et al. “Agent uml: A formalism for specifying multiagent interaction“. In *Agent-oriented software engineering*, pp. 91–103. Springer, Berlin, 2001.
- [18] G. S. Benato, F. J. Affonso, and E. Y. Nakagawa. “Infrastructure based on template engines for automatic generation of source code for self-adaptive software domain”. In *SEKE*, pp. 30–35, 2017.

- [19] C. A. Berry, J. Carnell, M. B. Juric, M. N. Kunnumpurath, N. Nashi, and S. Romanosky, “Chapter 5: Patterns Applied to Manage Security”, *J2EE Design Patterns Applied*.
- [20] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Çomak, and L. Karaçay, “Vulnerability prediction from source code using machine learning,” *IEEE Access*, vol. 8, pp. 150672–150684, 2020.
- [21] J. Bishop, “Language features meet design patterns: raising the abstraction bar”, *2nd International workshop on The role of abstraction in software engineering (ICSE’08)*, IEEE, pp. 1-7, Leipzig, Germany, 10-18 May 2008.
- [22] B. Blakley, C. Heath, and TheOpenGroup, Technical Guide. Security Design Patterns. 2004. <http://www.opengroup.org/>
- [23] M. J. Blas and S. Gonnet. “Computer-aided design for building multipurpose routing processes in discrete event simulation models”. *Engineering Science and Technology*, 24(1):22–34, 2021.
- [24] B. Boehm and V. R. Basili, “Software Defect Reduction Top 10 List,” *Computer* (Long. Beach. Calif.), vol. 34, pp. 135–137, 2001.
- [25] A. M. Braga, C. Rubira, and R. Dahab, “Tropyc: A Pattern Language for Cryptographic Software”. *5th Pattern Languages of Programming (PLoP’98) Conference*. 1998. Allerton Park, Illinois, USA.
- [26] L. C. Briand, Y. Labiche and A. Sauve, “Guiding the Application of Design Patterns Based on UML Models”, *22nd IEEE International Conference on Software Maintenance*, IEEE, pp. 234-243, Philadelphia, Pennsylvania, 24-27 September 2006
- [27] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, “Pattern-Oriented Software Architecture: A System of Patterns”. 1st Edition. 1996: *John Wiley & Sons*. 476 Pg.
- [28] S. Cao, X. Sun, L. Bo, Y. Wei, and B. Li, “BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection,” *Inf. Softw. Technol.*, vol. 136, p. 106576, 2021.
- [29] M. Carvalho, J. Demott, R. Ford, and D. A. Wheeler, “Heartbleed 101,” *IEEE Secur. Priv.*, vol. 12, no. 4, pp. 63–67, 2014.
- [30] K. Chander, K., and R. K. Sharma, “A Framework for Efficient Web Services Discovery” (Doctoral dissertation), 2017.
- [31] A. Chatzigeorgiou, A. Ampatzoglou, A. Ampatzoglou. And T. Amanatidis, “Estimating the breaking point for technical debt”, *7th International Workshop on Managing Technical Debt (MTD)*, Bremen, Germany, 2 October 2015.
- [32] F. Chen, C. Lu, H. Wu, and M. Li, „A semantic similarity measure integrating multiple conceptual relationships for web service discovery”. *Expert Systems with Applications*, 67, 19-31, 2017.
- [33] B. Chess and G. McGraw, “Static analysis for security,” *Secur. Privacy*, IEEE, vol. 2, pp. 76–79, 2004.
- [34] S. Chidamber, and C. Kemerer, "A metrics suite for object oriented design", *Transactions on Software Engineering*, IEEE Computer Society, 20 (6), pp. 476 – 493, 1994.
- [35] I. Chowdhury and M. Zulkernine, “Can Complexity, Coupling, and Cohesion Metrics Be Used As Early Indicators of Vulnerabilities?,” *Proc. 2010 ACM Symp. Appl. Comput.*, pp. 1963–1969, 2010.
- [36] M. Crasso, A. Zunino, and M. Campo, “Awsc: An approach to web service classification based on machine learning techniques”, *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 12(37), pp. 25-36, 2008.

- [37] R. T. Colombo, M. S. Pessôa, A. C. Guerra, A. B. Filho, and C. C. Gomes, “Prioritization of software security intangible attributes,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 37, no. 6, p. 1, 2012.
- [38] J.M. Conejero, R. Rodríguez-Echeverría, J. Hernandez, P.J. Clemente, C. Ortiz-Caraballo, E. Jurado, and F. Sanchez-Figueroa, “Early evaluation of technical debt impact on maintainability”, *Journal of Systems and Software*, 142, Elsevier, 2018, pp. 92–114.
- [39] W. Cunningham, “The WyCash Portfolio Management System”, *7th International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '92)*, ACM, Vancouver, Canada 18 – 22 October 1992.
- [40] CWE, “CWE - 2011 cwe/sans top 25 most dangerous software errors,” SANS Inst., p. 41, 2011.
- [41] H. K. Dam, T. Tran, T. T. M. Pham, S. W. Ng, J. Grundy, and A. Ghose, “Automatic feature learning for predicting vulnerable software components,” *IEEE Trans. Softw. Eng.*, vol. 14, no. 8, pp. 1–19, 2018.
- [42] H. K. Dam, T. Tran, and T. Pham, “A deep language model for software code,” arXiv Prepr. arXiv1608.02715, 2016.
- [43] S. Dang and P. Hamid Ahmad. “Text mining: Techniques and its application”. *International Journal of Engineering & Technology Innovations*, 1(4):866–2348, 2014.
- [44] F. Das Neves, and A. Garrido, “BodyGuard, in *Pattern Languages of Programs III*”, Addison-Wesley, Editor. 1998.
- [45] U. Dayanandan and V. Kalimuthu, “Software Architectural Quality Assessment Model for Security Analysis Using Fuzzy Analytical Hierarchy Process (FAHP) Method,” *3D Res.*, vol. 9, no. 3, 2018.
- [46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv Prepr. arXiv1810.04805, 2018.
- [47] R. Eisenberg, 2013, October. “Management of technical debt: a lockheed martin experience report”. *3rd International Workshop on Managing Technical Debt (MTD'13)*, Baltimore, USA.
- [48] M. Fang, D. Wang, Z. Mi, and M. S. Obaidat, „Web service discovery utilizing logical reasoning and semantic similarity”. *International Journal of Communication Systems*, 31(10), 2018.
- [49] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, “Security Testing: A Survey,” *Adv. Comput.*, vol. 101, pp. 1–51, 2016.
- [50] Z. Feng et al., “Codebert: A pre-trained model for programming and natural languages,” arXiv Prepr. arXiv2002.08155, 2020.
- [51] J. Ferber and O. Gutknecht. “A meta-model for the analysis and design of organizations in multi-agent systems”. In *Proceedings International Conference on Multi Agent Systems* (Cat. No. 98EX160), pp. 128–135, 1998.
- [52] D. Fensel, H. Lausen, A. Polleres, J. De Bruijn, M. Stollberg, D. Roman, and J. Domingue, “Enabling semantic web services: the web service modeling ontology”, Springer Science & Business Media, 2006.
- [53] D. G. Firesmith, “Specifying Reusable Security Requirements”. *Journal of Object Technology*, 2004. 3: p. 61-75.
- [54] R. Flanders, and E. B. Fernandez, “Data Filter Architecture Pattern”. *6th Conference on Pattern Languages of Programs, PLoP 1999*. 1999. Allerton Park, Monticello, Illinois.
- [55] S. Fogie, J. Grossman, R. Hansen, a. Rager, and P. D. Petkov, XSS Attacks: Cross Site Scripting Exploits and Defense. 2007.

- [56] E. Gamma, R. Helms, R. Johnson and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, *Addison-Wesley Professional*, Reading, MA, 1995.
- [57] I. Garcia-Magarino. “Towards the integration of the agent-oriented modeling diversity with a powertype-based language”. *Computer Standards & Interfaces*, 36(6):941–952, 2014.
- [58] E. Gelenbe et al., “NEMESYS: Enhanced network security for seamless service provisioning in the smart mobile ecosystem,” in *Information Sciences and Systems 2013*, Springer, 2013, pp. 369–378.
- [59] J. J. Gomez-Sanz, R. Fuentes, J. Pavón, and I. García-Magariño. “Ingenias development kit: a visual multi-agent system development environment”. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, pp. 1675–1676, 2008.
- [60] A. González-Briones, J. Prieto, F. De La Prieta, E. Herrera-Viedma, and J. M Corchado. “Energy optimization using a case-based reasoning strategy“. 18(3):865, 2018.
- [61] A. González-Briones, J. Prieto, F. De La Prieta, Y. Demazeau, and J. M Corchado. “Virtual agent organizations for user behaviour pattern extraction in energy optimization processes: A new perspective“. *Neurocomputing*, 2020.
- [62] M. Green and M. Smith, “Developers are Not the Enemy!: The Need for Usable Security APIs,” *IEEE Secur. Priv.*, vol. 14, 2016.
- [63] W. G. J. Halfond, J. Viegas, and A. Orso, “A Classification of SQL Injection Attacks and Countermeasures,” 2008.
- [64] I. Heitlager, T. Kuipers, and J. Visser, “A Practical Model for Measuring Maintainability,” *6th Int. Conf. Qual. Inf. Commun. Technol. (QUATIC 2007)*, pp. 30–39, 2007.
- [65] A. Hovsepian, R. Scandariato, W. Joosen, and J. Walden, “Software vulnerability prediction using text analysis techniques,” *Proc. 4th Int. Work. Secur. Meas. metrics - MetriSec '12*, p. 7, 2012.
- [66] D. Hovemeyer and W. Pugh, “Finding bugs is easy,” *ACM SIGPLAN Not.*, vol. 39, no. 12, p. 92, 2004.
- [67] M. Howard, D. LeBlanc, and J. Viega, *24 Deadly Sins of Software Security*. 2010.
- [68] N.L. Hsueh, P.H. Chu, P.A. Hsiung, M.J. Chuang, W. Chu, C.H. Chang, C.S. Koong and C.H. Shih, “Supporting Design Enhancement by Pattern-Based Transformation”, *34th Annual Computer Software and Applications Conference (COMPSAC '10)*, IEEE, pp. 462 – 467, Seoul , Korea, 19-23 July 2010.
- [69] ISO/IEC, “ISO/IEC 27001:2013(en) Information technology — Security techniques — Information security management systems — Requirements,” ISO/IEC, 2013.
- [70] ISO/IEC, “ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models,” ISO/IEC, 2011.
- [71] T. Ivanova and I. Batchkova. “Approach for model driven development of multi-agent systems for ambient intelligence”. *Artificial Intelligence in Industry 4.0: A Collection of Innovative Research Case-studies that are Reworking the Way We Look at Industry 4.0 Thanks to Artificial Intelligence*, pages 183–198, 2021.
- [72] A. Jazayeri and E. J. Bass. “Agent-oriented methodologies evaluation frameworks: A review”. *International Journal of Software Engineering and Knowledge Engineering*, 30(09):1337–1370, 2020.
- [73] S. Jörges. “Construction and evolution of code generators: A model-driven and service-oriented approach”, 7747. Springer, 2013.
- [74] N. Jovanovic, C. Kruegel, and E. Kirda, “Pixy: a static analysis tool for detecting Web application vulnerabilities,” *2006 IEEE Symposium on Security and Privacy (S&P'06)*, 2006, p. 6 pp.-263.

- [75] G. Kakarontzas, E. Constantinou, A. Ampatzoglou, and I. Stamelos, “Layer assessment of object-oriented software: A metric facilitating white-box reuse”, *Journal of Systems and Software*, 86 (2), pp. 349-366.
- [76] I. Kalouptsoglou, M. Siavvas, D. Tsoukalas, and D. Kehagias, Cross-Project Vulnerability Prediction Based on Software Metrics and Deep Learning, vol. 12252 LNCS. 2020.
- [77] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes. “Big code!= big vocabulary: Open-vocabulary models for source code”. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1073–1085. IEEE, 2020.
- [78] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyevev, V. Fedak, and A. Shapochka, “A case study in locating the architectural roots of technical debt”, *37th International Conference on Software Engineering (ICSE) 2*, IEEE/ACM, Florence, Italy, May 2015, pp. 179–188.
- [79] K. Keogh and L. Sonenberg. „Designing multi-agent system organisations for flexible runtime behaviour”. *Applied Sciences*, 10(15):5335, 2020.
- [80] B. Keepence and M. Mannion, “Using Patterns to Model Variability in Product Families”, *IEEE Software*, IEEE, 16 (4), pp. 102-108, July 1999.
- [81] C. King, E. Osmanoglu, and C. Dalton, “Security Architecture”. 2001: McGraw-Hill.
- [82] M. Kis, “Information Security Antipatterns in Software Requirements Engineering”. *9th Conference on Pattern Languages of Programs (PLoP'2002)*. 2002.
- [83] I. Krsul, “Software Vulnerability Analysis,” Department of Computer Sciences, Purdue University, 1998.
- [84] P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” *IEEE Software*, vol. 29, no. 6, pp. 18–21, Nov. 2012.
- [85] D. E Krutz, M. Mirakhorli, S. A Malachowsky, A. Ruiz, J. Peterson, A. Filipiski, and J. Smith. “A dataset of opensource android applications”. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 522–525. IEEE, 2015.
- [86] M. Kumar, R. Bhatia, and D. Rattan. “A survey of web crawlers for information retrieval”. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(6):e1218, 2017.
- [87] S. T. Lai, “An analyzer-based software security measurement model for enhancing software system security,” *2nd WRI World Congr. Softw. Eng. WCSE 2010*, vol. 2, pp. 93–96, 2010.
- [88] J. F. Lee Brown, J. DiVietri, G. Diaz de Villegas, and E. B. Fernandez, “The Authenticator Pattern”. *6th Conference on Pattern Languages of Programs, PLoP 1999*. 1999. Allerton Park, Monticello, Illinois.
- [89]] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management,” *Journal of System and Software*, vol. 101, pp. 193– 220, Mar. 2015.
- [90] W. Li, and S. Henry, "Object-oriented metrics that predict maintainability", *Journal of Systems and Software*, Elsevier, 23 (2), pp. 111 – 122, 1993.
- [91] Z. Li et al., “VulDeePecker: A Deep Learning-Based System for Vulnerability Detection,” Jan. 2018.
- [92] F. Liu, P. Li, and D. Deng, “Device-oriented automatic semantic annotation in IoT”. *Journal of Sensors*, 2017.
- [93] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra. “Aroma: Code recommendation via structural code search”. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–28, 2019.

- [94] J. Luszcz, “Apache Struts 2: how technical and development gaps caused the Equifax Breach,” *Netw. Secur.*, vol. 2018, no. 1, pp. 5–8, Jan. 2018.
- [95] S. MacDonald, D. Szafron, J. Schaeffer, J. Anvik, S. Bromling and K. Tan, “Generative Design Patterns”, *17th IEEE international conference on Automated software engineering*, pp. 23, Edinburgh, UK, 23-27 September 2002
- [96] S. MacDonald, K. Tan, J. Schaeffer and D. Szafron, “Deferring Design Pattern Decisions and Automating Structural Pattern Changes Using a Design-Pattern-Based Programming System”, *Transactions on Programming Languages and Systems*, ACM, 31(3), article 9, April 2009.
- [97] A. MacCormack, and D.J. Sturtevant, “Technical debt and system architecture: The impact of coupling on defect-related activity”, *Journal of Systems and Software* 120 (2016) 170–182. Elsevier.
- [98] V. Markovtsev and W. Long. “Public git archive: a big code dataset for all”. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 34–37, 2018.
- [99] G. McGraw and G. McGraw, “Why Building Secure,” no. October, pp. 229–238, 2000.
- [100] G. McGraw, “Software Security: Building Security” In. *Addison-Wesley Professional*, 2006.
- [101] I. Medeiros, N. Neves, and M. Correia, “Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining,” *IEEE Trans. Reliab.*, vol. 65, no. 1, 2016.
- [102] N. Medeiros, N. Ivaki, P. Costa, and M. Vieira, “An Approach for Trustworthiness Benchmarking Using Software Metrics,” in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2018, pp. 84–93.
- [103] M. Meyer, “Pattern-based Reengineering of Software Systems”, *13th Working Conference on Reverse Engineering*, pp.305-306, Benevento, Italy, 23-27 October 2006
- [104] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, “Exploring Software Security Approaches in Software Development Lifecycle: A Systematic Mapping Study,” *Comput. Stand. Interfaces*, vol. 50, no. October 2016, pp. 107–115, 2016.
- [105] P. Morrison, K. Herzig, B. Murphy, and L. Williams, “Challenges with Applying Vulnerability Prediction Models,” *Proc. 2015 Symp. Bootcamp Sci. Secur.*, p. 4:1-4:9, 2015.
- [106] N. Munaiah, F. Camilo, W. Wigham, A. Meneely, and M. Nagappan, “Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project,” *Empir. Softw. Eng.*, vol. 22, no. 3, pp. 1305–1347, 2017.
- [107] H. Nabli, R. Ben Djemaa, and I. A. Ben Amor. “Efficient cloud service discovery approach based on lda topic modelling”. *Journal of Systems and Software*, 146:233–248, 2018.
- [108] T. R. G. Nair and R. Selvarani. 2010. “Estimation of software reusability: an engineering approach”. *SIGSOFT Softw. Eng. Notes* 35, 1 (January 2010), 1-6.
- [109] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, “Predicting vulnerable software components,” *Proc. 14th ACM Conf. Comput. Commun. Secur. CCS 07*, p. 529, 2007.
- [110] M. O’Cinneide, “Automated refactoring to introduce design patterns”, *Proceedings of the 22nd international conference on Software engineering (ICSE’00)*, IEEE, pp.722-724, Limeric, Ireland, 4-11 June 2000
- [111] M. O’Cinneide and P. Nixon, “A Methodology for the Automated Introduction of Design Patterns”, *Proceedings of the 15th IEEE International Conference on Software Maintenance*, IEEE, pp. 463, Oxford, England, 30 August – 03 September 1999

- [112] M. O' Cinneide and P. Nixon, "Automated software evolution towards design patterns", *Proceedings of the 4th International Workshop on Principles of Software Evolution (ICSE'01)*, IEEE, pp.162-165, Vienna, Austria, 12-19 May 2001
- [113] N. Oldham, C. Thomas, A. Sheth, and K. Verma, "Meteor-s web service annotation framework with machine learning classification", *In International Workshop on Semantic Web Services and Web Process Composition*, Berlin, Heidelberg, 2014.
- [114] Owasp, "OWASP Top 10 - 2013," OWASP Top 10, p. 22, 2013.
- [115] Y. Pang, X. Xue, and H. Wang, "Predicting Vulnerable Software Components through Deep Neural Network," *Proc. 2017 Int. Conf. Deep Learn. Technol. - ICDLT '17*, pp. 6–10, 2017.
- [116] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, B. J. Krämer, "05462 service-oriented computing: A research roadmap". *In Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2006.
- [117] D.L. Parnas, 1994, May. "Software aging". *16th International Conference on Software Engineering* (pp. 279-287). IEEE.
- [118] V. M. Patro, and M. R. Patra, "Classification of Web services using fuzzy classifiers with feature selection and weighted average accuracy". *Transactions on Networks and Communications*, 3(2), 107, 2015.
- [119] J. Pavón, J. J. Gómez-Sanz, and R. Fuentes. "The ingenias methodology and tools". *In Agent-oriented methodologies*, pp. 236–276. IGI Global, 2005.
- [120] Pfleeger and S. L. Pfleeger, *Security in Computing*, vol. 2nd, no. 1. 2006.
- [121] H. S. Pham, S. Nijssen, K. Mens, D. Di Nucci, T. Molderez, C. De Roover, J. Fabry, and V. Zaytsev. "Mining patterns in source code using tree mining algorithms". *In International Conference on Discovery Science*, pp. 471–480. Springer, 2019.
- [122] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of GitHub README files". *Empirical Software Engineering*, 24(3), 1296-1327, 2019
- [123] R. Priya, X. Wang, Y. Hu, and Y. Sun. "A deep dive into automatic code generation using character based recurrent neural networks". *In 2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 369–374. IEEE, 2017.
- [124] L. Purohit, and S. Kumar, "A classification based web service selection approach. IEEE Transactions on Services Computing, 2018.
- [125] F. Rademacher, J. Sorgalla, P. Wizenty, S. Sachweh, and A. Zündorf. "Graphical and textual model-driven microservice development". *In Microservices*, pp. 147–179. Springer, 2020.
- [126] J. Ramachandran, "Designing Security Architecture Solutions". 2002: *John Wiley & Sons*.
- [127] M. Riaz, E. Mendes, and E. Tempero. 2009. "A systematic review of software maintainability prediction and metrics". *In Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, Florida, USA, 367-377.
- [128] A. Rivas, A. González-Briones, J. J Cea-Morán, A. P. Pérez, and J. M Corchado. "My-trac: System for recommendation of points of interest on the basis of twitter profiles". *Electronics*, 10(11):1263, 2021.
- [129] A. Rivas, A. Gonzalez-Briones, G. Hernandez, J. Prieto, and P. Chamoso. "Artificial neural network analysis of the academic performance of students in virtual learning environments". *Neurocomputing*, 423:713–720, 2021.
- [130] S. Romanosky, "Enterprise Security Patterns". *7th European Conference on Pattern Languages of Programs (EuroPlop'02)*. 2002. Irsee, Germany.

- [131] S. Romanosky, “Security Design Patterns”. 2001.
- [132] D. Rosado, C. Gutiérrez, E. Fernández-Medina, and M. Piattini, 2006. “Security Patterns Related to Security Requirements”. *WOSIS*.
- [133] F. Röser, 2012. “Security Design Patterns in Software Engineering - Overview and Example”.
- [134] R. Russell et al., “Automated vulnerability detection in source code using deep representation learning,” in *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, 2018, pp. 757–762.
- [135] P. Salini and S. Kanmani, “Survey and analysis on security requirements engineering,” *Comput. Electr. Eng.*, vol. 38, no. 6, pp. 1785–1797, 2012.
- [136] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, “Predicting vulnerable software components via text mining,” *IEEE Trans. Softw. Eng.*, vol. 40, no. 10, pp. 993–1006, 2014.
- [137] M. Schumacher, M. and U, Roedig, “Security Engineering with Patterns”. *8th Conference on Patterns Languages of Programs, PLoP 2001*. 2001. Monticello, Illinois, USA.
- [138] A. Sharma, P. S. Grover, and R. Kumar. 2009. “Reusability assessment for software components”. *SIGSOFT Softw. Eng. Notes* 34, 2 (February 2009), 1-6.
- [139] Y. Shin and L. Williams, “Is complexity really the enemy of software security?,” *4th ACM workshop on Quality of protection*, 2008, pp. 47–50.
- [140] Y. Shin and L. Williams, “An empirical model to predict security vulnerabilities using code complexity metrics,” *2nd ACM-IEEE international symposium on Empirical software engineering and measurement*, 2008, pp. 315–317.
- [141] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, “Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities,” *IEEE Trans. Softw. Eng.*, vol. 37, no. 6, pp. 772–787, 2011.
- [142] M. Siavvas, D. Kehagias, D. Tzovaras, and E. Gelenbe, “A hierarchical model for quantifying software security based on static analysis alerts and software metrics,” *Softw. Qual. J.*, 2021.
- [143] M. Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias, and D. Tzovaras, “Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises,” *Enterp. Inf. Syst.*, pp. 1–43, Sep. 2020.
- [144] C. Steel, R. Nagappan, and R. Lai, “Core Security Patterns”. 2005: Prentice Hall PTR. 1088 Pg.
- [145] E. Sunitha and P. Samuel. “Automatic code generation from uml state chart diagrams”. *IEEE Access*, 7:8591–8608, 2019.
- [146] TabNine, Autocompletion with deep learning. <https://www.kite.com/>, 2019.
- [147] P. Tonella and G. Antoniol, “Object Oriented Design Pattern Inference”, *Journal of Software Maintenance*, Wiley, 13 (5), September-October 2001
- [148] B. Uyanik and V. H. ŞAHİN. “A template-based code generator for web applications”. *Turkish Journal of Electrical Engineering & Computer Sciences*, 28(3):1747–1762, 2020.
- [149] H. van Vliet, "Software Engineering: Principles and Practice (3rd Edition)", *Wiley & Sons*, Chichester, England, 1993.
- [150] A. Vaswani et al., “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.
- [151] S. Wagner et al., “Operationalised product quality models and assessment: The Quamoco approach,” *Inf. Softw. Technol.*, vol. 62, pp. 101–123, 2015.
- [152] J. Walden, J. Stuckman, and R. Scandariato, “Predicting vulnerable components: Software metrics vs text mining,” *Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 23–33, 2014.

- [153] H. Washizaki, H. Yamamoto, and Y. Fukazawa. "A metrics suite for measuring reusability of software components", *9th International Software Metrics Symposium*, IEEE, 2003.
- [154] R. Wassermann, "Using Security Patterns to Model and Analyze Security Requirements". 2004. p. 151.
- [155] R. Waszkowski. "Low-code platform for automating business processes in manufacturing". *IFAC-PapersOnLine*, 52(10):376–381, 2019.
- [156] R. J. Wieringa, "Design science methodology for information systems and software engineering", Springer, 2014.
- [157] G. Wurster and P. C. van Oorschot, "The developer is the enemy," *NSPW '08 Proc. 2008 Work. New Secur. Paradig.*, pp. 89–97, 2008.
- [158] Y. Yang, W. Ke, W. Wang, and Y. Zhao, "Deep learning for web services classification", In *2019 IEEE International Conference on Web Services (ICWS)* (pp. 440-442), 2019.
- [159] J. Yang, D. Ryu, and J. Baik, "Improving vulnerability prediction accuracy with Secure Coding Standard violation measures," *2016 Int. Conf. Big Data Smart Comput. BigComp 2016*, pp. 115–122, 2016.
- [160] S. S. Yau and N. Dong, "Integration in Component-Based Software Development Using Design Patterns", *24th International Computer Software and Applications Conference (COMPSAC'00)*, IEEE, pp.369, Taipei, Taiwan, 25-28 October 2000
- [161] J. Yin, M. Tang, J. Cao, and H. Wang, "Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description," *Knowledge-Based Syst.*, vol. 210, p. 106529, 2020.
- [162] J. Yli-Huumo, A. Maglyas, and K. Smolander, "How do software development teams manage technical debt?—An empirical study", *Journal of Systems and Software*, 120 (2016) 195–218. Elsevier.
- [163] J. Yoder, and J. Barcalow, "Architectural Patterns for Enabling Application Security". *4th Conference on Patterns Language of Programming, PLOp 1997*. 1997. Monticello, Illinois, USA.
- [164] L. Yuan-jie, and C. Jian, "Web service classification based on automatic semantic annotation and ensemble learning", In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum* (pp. 2274-2279). IEEE, 2012.
- [165] S. Zafar, M. Mehboob, A. Naveed, and B. Malik, "Security quality model: an extension of Dromey's model," *Softw. Qual. J.*, vol. 23, no. 1, 2015.
- [166] F. Zambonelli and A. Omicini. "Challenges and research directions in agent-oriented software engineering". *Autonomous agents and multi-agent systems*, 9(3):253–283, 2004.
- [167] S. Zatout, M. Boufaida, M. S. Benabdelhafid, and M. L. Berkane. "A model-driven approach for the verification of an adaptive service composition". *International Journal of Web Engineering and Technology*, 15(1):4–31, 2020.
- [168] N. Zazworka, A. Vetró, C. Izurieta, S. Wong, Y. Cai, C. Seaman and F. Shull., "Comparing four approaches for technical debt identification", *Software Quality Journal*, 22 (3), pp. 403–426, 2014
- [169] N. Zhang, J. Wang, Y. Ma, K. He, Z. Li, and X. F. Liu, "Web service discovery based on goal-oriented query expansion". *Journal of Systems and Software*, 142, 73-91. 2018.
- [170] M. Zhang, X. de C. de Carnavalet, L. Wang, and A. Ragab, "Large-Scale Empirical Study of Important Features Indicative of Discovered Vulnerabilities to Assess Application Security," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 9, pp. 1–1, 2019.

- [171] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun, and S. Li, “Combining software metrics and text features for vulnerable file prediction,” *20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2015, pp. 40–49.
- [172] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. a S. E. I. T. on Vouk, “On the value of static analysis for fault detection in software,” *Softw. Eng. IEEE Trans.*, vol. 32, no. 4, pp. 240–253, 2006.
- [173] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, “Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks,” *arXiv Prepr. arXiv1909.03496*, 2019.